

Spis treści

Obsługa interpretera WSH.....	2
Windows Script Host (WSH).....	2
Obsługa WSH.....	2
Specyfikacja środowiska WSH.....	4
Zadania.....	5
Podstawy składni Visual Basic.....	5
Instrukcja VB.....	5
Zmienne.....	6
Operatory.....	6
Procedury.....	7
Funkcje.....	8
Zadania.....	8
Komunikacja z użytkownikiem.....	9
Zadania.....	9
Struktury języka Visual Basic.....	10
Pętla For.....	10
Warunek If.....	10
Pętla Do.....	12
Zadania.....	13
Obiekty standardowe w WSH.....	13
Obiekt WScript.....	13
Obiekt WshArguments.....	14
Obiekt WshEnvironment.....	14
Obiekt WshNetwork.....	15
Obiekt WshShell.....	17
Zadania.....	18
Wybrane rozszerzenia WSH.....	18
Obsługa strumieni.....	18
Strumień StdOut.....	18
Strumień StdIn.....	19
Obsługa plików i folderów.....	20

Inne zastosowania WSH.....	21
Zadania.....	21

Obsługa interpretera WSH

Windows Script Host (WSH)

Microsoft® Windows® Script Host jest narzędziem pozwalającym na wykonywanie ciągów poleceń (skryptów) napisanych w wielu językach. Standardowo system Windows dostarcza języki VBScript oraz Jscript, ale dostępne są także: PerlScript, TCL, REXX i Python, dostarczane przez inne firmy. WSH jest idealny do pisania skryptów administracyjnych, skryptów logowania oraz wszelkiej automatyzacji pracy. Dzięki współpracy z technologią ActiveX pozwala na automatyzację wielu czynności samego środowiska Windows, jak i aplikacji dla niego przeznaczonych. Do najważniejszych zalet WSH należą:

- możliwość uruchamiania zarówno przy pomocy środowiska graficznego, jak i linii poleceń,
- możliwość jednoczesnego uruchomienia wielu skryptów (przy pomocy jednego wywołania pliku .WSF),
- niskie wymagania w odniesieniu do ilości pamięci operacyjnej,
- rozpoznawanie użytego języka na podstawie rozszerzenia pliku.

Środowisko skryptów WSH jest zorientowane obiektowo. Dostarcza elementów, obiektów, metod i własności pozwalających na realizację następujących zadań:

- wyświetlanie wiadomości na ekranie,
- operacje na obiektach,
- obsługa dysków sieciowych,
- obsługa drukarek sieciowych,
- pobieranie i modyfikowanie zmiennych środowiskowych,
- dostęp do rejestru systemowego.

Obsługa WSH

Skrypty WSH można pisać przy pomocy dowolnego edytora tekstowego lub zewnętrznego środowiska IDE. System Windows nie dostarcza wyspecjalizowanych narzędzi do edycji skryptów. Rozszerzenia plików zawierających skrypty wskazują na język zapisu:

- VBS – Visual Basic Script,
- JS – Java Script,
- WSF – format swobodny ze specyfikacją XML, mogący zawierać skrypty w dowolnych językach.

Aby pokazać obsługę narzędzi WSH, musimy przygotować pierwszy skrypt. W tym celu należy utworzyć (np. przy pomocy notatnika) plik *Hello.vbs* i umieścić w nim kod w Visual Basicu o treści:

```
WScript.Echo "Hello World!"
```

lub utworzyć plik *Hello.js* o treści:

```
WScript.Echo("Hello World!");
```

Do uruchamiania skryptów służą dwa polecenia: WScript (wersja WSH z interfejsem graficznym oraz CScript (wersja przeznaczona do uruchamiania skryptów z linii poleceń). Oba polecenia mają podobną składnię. Podstawowa różnica polega na tym, że CScript wyświetla komunikaty na konsoli, zaś WScript pokazuje je w postaci okienek.

Składnia polecenia:

```
CScript|Wscript nazwa_skryptu.rozszerzenie [opcja...] [argumenty...]
```

//B	Tryb wsadowy - pomija wyświetlanie błędów i monitów w skryptach
//D	Włącza debugowanie aktywne
//E:aparat	Wykonuje skrypt przy użyciu aparatu
//H:CScript	Ustawia program CScript.exe jako domyślny host skryptów
//H:Wscript	Ustawia program WScript.exe jako domyślny host skryptów (ta wartość jest domyślna)
//I	Tryb interaktywny (domyślny, przeciwieństwo trybu //B)
//Job:xxxx	Wykonuje zadanie WSF
//Logo	Wyświetla logo (domyślnie)
//Nologo	Nie wyświetla logo: podczas wykonywania skryptu nie będzie wyświetlany transparent
//S	Zapisuje bieżące opcje wiersza poleceń dla tego użytkownika
//T:nn	Limit czasu w sekundach: maksymalny dozwolony czas wykonywania skryptu
//X	Wykonuje skrypt w debuggerze
//U	Używa standardu Unicode dla przekierowań We/Wy z konsoli (tylko przy komendzie CScript)

Przykłady:

```
CScript test1.vbs //nologo
```

Wykonuje skrypt nie wyświetlając na konsoli tekstu powitalnego (informującego o wersji WSH), zaś informacje o błędach wyświetla na konsoli.

```
CScript test1.vbs //T:10
```

Polecenie wykona skrypt i zatrzyma go, jeżeli czas pracy przekroczy 10 sekund.

Uruchomienie polecenia WScript bez wyspecyfikowania parametrów powoduje wyświetlenie okna pozwalającego na podstawową konfigurację WSH.

Specyfikacja środowiska WSH

Pliki z rozszerzeniem WSF pozwalają na definiowanie złożonych skryptów używających wielu języków. Strukturę takiego pliku definiują elementy XML. Najważniejsze z nich przedstawiono poniżej:

<job> Pozwala na określenie nazwy oraz zakresu pojedynczego zadania. Składnia:

```
<job [id=JobID]>  
kod zadania </job>
```

<script> Specyfikuje język w którym napisany jest fragment skryptu. Składnia:

```
<script language="language"  
[src="strFile"]>   treść skryptu  
</script>
```

<package> Definiuje pakiet zadań. Nie jest wymagany w przypadku skryptu jednozadaniowego. Składnia:

```
<package>  
    kod jednego lub więcej zadań  
</package>
```

Przykład:

```
<package>  
  <job id="DoneInVBS">  
    <?job debug="true"?>  
    <script language="VBScript">  
      WScript.Echo "This is VBScript"  
    </script>  
  </job>  
  
  <job id="DoneInJS">  
    <?job debug="true"?>  
    <script language="JScript">  
      WScript.Echo("This is JScript");  
    </script>  
  </job>  
</package>
```

Gdy skrypt definiuje wiele zadań, wykonywane zadanie wybierane jest przez parametr w linii poleceń interpretera.

Zadania

- Jakie języki dostarczane są z WSH?
- W jaki sposób interpreter rozpoznaje zastosowany język?
- Co zawierają pliki WSF?
- Czym różnią się interpretery WScript i CScript?

Podstawy składni Visual Basic

Poniższa instrukcja zawiera jedynie najważniejsze informacje o języku Visual Basic (VB). Szczegółową dokumentację możecie Państwo odnaleźć na stronie MSDN:

<http://msdn2.microsoft.com/en-us/library/d1wf56tt.aspx>

Instrukcja VB

Skrypt WSH napisany w VB jest ciągiem instrukcji (programem). Pisanie programów dla WSH nie różni się niczym od innych języków programowania. Poniżej postaram się zwrócić uwagę na najważniejsze różnice pomiędzy VB a popularnymi językami programowania. W VB każda instrukcja znajduje się w oddzielnej linii. Oznacza to, że w przeciwieństwie do innych języków, w których instrukcje oddziela określony separator, nie możemy dowolnie przenosić linii. Jeśli wymagane jest przeniesienie części długiej instrukcji do następnej linii należy umieścić na końcu niedokończonej instrukcji znak "_":

początek długiej instrukcji _ oraz jej dokończenie

W przeciwieństwie do wielu poprawnych języków, VB nie jest czuły na wielkość liter - nazwa zapisana małymi literami będzie równoważna napisanej wielkimi:

NowaZmienna NOWAZMIENNA nowazmienna

Komentarze, czyli linie nieinterpretowane, należy rozpoczynać znakiem "'".

Najważniejszymi składnikami języka VB są:

- zmienne,
- operatory,
- procedury,
- funkcje.

Zmienne

Zmienne z poprzedniego punktu był zmiennymi typu Variant, a więc mogące przechowywać dowolny typ danych. W VB nie ma potrzeby jawnego deklarowania zmiennych. Zmienna zostaje utworzona w chwili jej pierwszego przypisania. Istnieje jednak możliwość jawnej deklaracji zmiennej. Służy do tego słowo kluczowe *Dim*:

```
Dim NazwaZmiennej
```

Wersja VB dostępna w WSH ma ograniczone możliwości definiowania typów zmiennych. W przeciwieństwie do innych implementacji VB nie można jawnie określić typu danych przy pomocy słowa *As*.

VisualBasic pozwala także na deklaracje tablic czyli zbiorów zmiennych tego samego typu. Aby zadeklarować tablicę wpisujemy:

```
Dim nazwa_tablicy (rozmiar)
```

Odwołanie do elementów tablicy można wykonać następująco:

```
nazwa_tablicy (3) = 45
```

Dla tablicy zadeklarowanej `Dim tablica(5)` można się odwoływać indeksami od 0 do 4. Inny zakres indeksów można uzyskać operatorem `to` np.:

```
Dim tablica (5 to 10)
```

Operatory

Operatory to składniki języka programowania wykonujące proste operacje arytmetyczne, logiczne i inne. Zazwyczaj mają postać jednego lub pary znaków. Najczęściej używanym operatorem jest przypisanie `=`. W VB ma ono postać:

```
zmienna = wartość
```

Najważniejszymi operatorami arytmetycznymi w VB są:

+	dodawanie
/	dzielenie

*	mnożenie
^	potęgowanie
\	dzielenie całkowite
mod	modulo (reszta z dzielenia)

Najważniejszymi operatorami logicznymi w VB są:

=, >, <, >=>, <=<	porównania
and	koniunkcja
or	alternatywa
not	negacja
xor	alternatywa wykluczająca

Inne ważne operatory to:

=	przypisanie
&	łączenie łańcuchów znaków

Procedury

Za funkcjonalność danego programu odpowiadają wywołane w nim procedury. Wywołanie procedury w VB może mieć trzy różne składnie:

```
nazwa_procedure(wartość)
```

lub

```
nazwa_procedure wartość
```

lub

```
nazwa_procedure nazwa_parametru:=wartość
```

Cechą charakterystyczną VB jest możliwość deklarowania parametrów domyślnych, których nie trzeba podawać podczas wywołania. Jeśli chcemy pominąć parametr po prostu nie podajemy je do wartości. Np.:

```
procedura(wartość1, , wartość3)
```

lub

```
procedura parametr10:=wartość
```

Funkcje

Jedyna różnica pomiędzy procedurą a funkcją jest taka, iż funkcja przekazuje wartość (może pełnić rolę r-wyrażenia). Wywołanie funkcji może mieć postać:

```
zmienna = funkcja(parametr)
```

VB w WSH dysponuje szeregiem funkcji i procedur standardowych. Najważniejsze z nich to:

CDate()	Przekształca wartość w datę
CInt()	Przekształca wartość w liczbę całkowitą
CStr()	Przekształca wartość w ciąg znaków
Date()	Zwraca datę systemową
DateDiff()	Wyznacza okres pomiędzy dwiema datami
DatePart()	Wyznacza część daty (rok, miesiąc, dzień, etc.)
GetObject()	Zwraca wskaźnik na obiekt automatyzacji (szczegóły dalej)
InputBox()	Pobiera dane od użytkownika
InStr()	Odszukuje dany ciąg w innym ciągu znaków
LCase()	Zmienia na małe litery
Left()	Pobiera zadaną liczbę znaków z lewej strony ciągu
Len()	Podaje długość ciągu znaków
MsgBox()	Wyświetla komunikat lub dialog z przyciskami
Now()	Zwraca bieżący czas i datę
Replace()	Zamienia podciąg znaków na inny
Right()	Pobiera zadaną liczbę znaków z prawej strony ciągu
Split()	Ciąg znaków podzielonych określonym separatorem zamienia w tablicę
Time()	Zwraca bieżący czas
UCase()	Zamienia na wielkie litery

Zadania

- Jaki znak jest separatorem instrukcji w VB?
- Jakie typy liczbowe występują w VB?
- Wymień funkcje i procedury VB operujące na dacie i czasie.
- Jakich operatorów użyjesz, aby sprawdzić podzielność dwóch liczb?

Komunikacja z użytkownikiem

Większość programów posiada jakiś interfejs użytkownika. W najprostszym przypadku zbudowany jest on z instrukcji pobierających tekst od użytkownika oraz wyświetlający komunikaty. W WSH do dyspozycji mamy instrukcje wyświetlające proste okna dialogowe.

Aby wyświetlić tekst na ekranie należy skorzystać z instrukcji MsgBox:

```
MsgBox "Wyświetlany tekst"
```

Do wprowadzania danych do programu służy instrukcja InputBox:

```
zmienna = InputBox("Tekst zachęty: ")
```

Przykład wykorzystania obu poleceń:

```
Dim sMyName  
sMyName = InputBox("Type your name") MsgBox "Hello,  
" & sMyName
```

Innym sposobem wyświetlania komunikatów jest polecenie:

```
WScript.Echo "Hello World"
```

W przeciwieństwie do MsgBox działanie metody jest zróżnicowane w zależności od interpretera w którym został skrypt uruchomiony. W CScript zostanie wyświetlony w postaci tekstu na konsoli, zaś w Wscript pokazane zostanie okno z komunikatem.

Istnieje także możliwość wyświetlenia okienka z zapytaniem wyposażonego w przyciski umożliwiające odpowiedź. Służy do tego wymieniona wcześniej instrukcja MsgBox. Aby wyświetliła ona zestaw przycisków należy podać jej drugi parametr:

```
iResponse = MsgBox("Czy kontynuować?", 4)
```

Szczegóły dotyczące wartości drugiego parametru można znaleźć w dokumentacji.

Zadania

- Napisz program obliczający wynik dodawania dwóch liczb wprowadzonych przez użytkownika.

Struktury języka Visual Basic

Pętla For

Pętla For nazywana jest również pętlą For-Next. Pozwala ona na wielokrotne powtarzanie bloku kodu. W przeciwieństwie do pętli typu Do wykonuje to ściśle określoną ilość razy. Powtórzenia pętli zwane są również iteracjami pętli.

Składnia instrukcji For:

```
For ZmiennaLicznik = WartośćPoczątkowa To WartośćKońcowa [Step  
WartośćKroku] Blok instrukcji VB wykonywany Next
```

Wyrażenia w nawiasach [] są opcjonalne i nie muszą występować w instrukcji. W przypadku, gdy pominięta zostanie część rozkazu określająca WartośćKroku, Visual Basic przyjmie domyślną WartośćKroku=1.

ZmiennaLicznik musi być zmienną (nie może być np. elementem sterującym). Jest ona wykorzystywana przez pętlę jako licznik powtórzeń.

Program wyświetlający liczby od 1 do 10 będzie miał postać:

```
Dim i  
For i = 1 To 10  
    WScript.Echo (CStr(i) + " ") Next
```

W kodzie dodatkowo pojawiła się instrukcja przekształcająca zmienną typu liczbowego w zmienną tekstową CStr. Operację odwrotną można wykonać funkcjami CInt lub CDBl. Poza tym wykonana jest operacja dodawania na dwóch wyrażeniach tekstowych. Operacja ta powoduje połączenie dwóch ciągów.

Warunek If

If jest najprostszą instrukcją wyboru. Dokonuje ona sprawdzenia podanego warunku i jeżeli jest on prawdziwy (ma wartość TRUE) wykonywany jest podany blok rozkazów. W przypadku gdy warunek jest fałszywy blok ten jest omijany.

Składnia instrukcji If:

```
If Warunek Then  
    Blok instrukcji VB wykonywany gdy Warunek jest prawdziwy  
End If
```

Przykładem wykorzystania If może być:

```
imie = InputBox("Jak się nazywasz?")

If imie = "Hubert" Then
    MsgBox ("Witaj Hubert")
End If
```

Instrukcja If wykonuje część programu tylko w przypadku, gdy sprawdzany warunek ma wartość TRUE. Nie wykonuje go jednak w przypadku, gdy ma on wartość FALSE. Rozbudowaną wersją instrukcji If jest If ...Else. W przypadku gdy sprawdzany warunek ma wartość TRUE wykonuje ona blok rozkazów zawarty między Then i Else, gdy zaś warunek ma wartość FALSE wykonuje blok rozkazów zawarty między Else i End If. Może ona więc "reagować" zarówno jeżeli warunek jest spełniony, jak i gdy nie jest spełniony.

Instrukcja If...Else nazywana jest wyrażeniem wzajemnie wykluczającym się, ponieważ może zostać wykonany tylko jeden z zawartych w nim bloków rozkazów.

Składnia instrukcji If...Else:

```
If Warunek Then
    Blok instrukcji VB wykonywany gdy Warunek jest prawdziwy
Else
    Blok instrukcji VB wykonywany gdy Warunek nie jest
    prawdziwy End If
```

Przykładem wykorzystania If..Else może być:

```
wynik = MsgBox("Wszystko w porządku?", vbYesNo)

If wynik = vbYes Then
    WScript.Echo ("Użytkownik czuje się dobrze")
Else
    WScript.Echo ("Użytkownik czuje się źle")
End If
```

Możliwość zagnieżdżania instrukcji If..Else (umieszczania jednej instrukcji If wewnątrz bloku rozkazów innej instrukcji If) wymusiło powstanie konstrukcji If...ElseIf. Jest to rozbudowana wersja instrukcji If ...Else. Pozwala ona na kolejne sprawdzanie wielu warunków.

Składnia instrukcji If...ElseIf:

```
If Warunek1 Then
    Blok instrukcji
ElseIf Warunek2 Then
    Blok instrukcji
Else
```

Blok instrukcji End If

Pętla Do

Pętla typu Do While jest chyba najbardziej powszechną pętlą Visual Basic. Wymaga ona zastosowania wyrażenia porównania. Wykorzystuje do tego celu operatory porównania. Pętla ta jest ograniczona słowami kluczowymi Do i Loop. W ciele pętli może znajdować się jedna instrukcja lub cały blok instrukcji VB. Blok ten jest wykonywany tak długo jak długo podany warunek jest prawdziwy. Ważne jest więc, aby w ciele pętli umieścić instrukcje, które doprowadzą do tego, że podany warunek kiedyś stanie się fałszywy. W przeciwnym razie pętla się nie zakończy.

Składnia instrukcji Do While:

Do While (Warunek) Blok instrukcji VB wykonywany gdy Warunek jest prawdziwy Loop
--

Blok instrukcji umieszczony w ciele pętli wykonywany jest tak długo, jak długo warunek pętli jest prawdziwy. Kiedy tylko stanie się fałszywy, to pętla kończy swoje działanie i przechodzi do rozkazu następnego za pętlą (za Loop). Jeżeli warunek jest fałszywy już na samym początku przed wykonaniem pętli, to blok instrukcji w ciele pętli nie zostanie wykonany ani razu. Charakterystyczne dla pętli Do While jest to, że jej ciało może nie zostać wykonane ani razu. Nieco inaczej działa pętla Do...Loop While, której ciało jest zawsze wykonywane przynajmniej jeden raz.

Najczęściej w Warunku pętli wykorzystywana jest jakaś zmienna. Należy zadbać, aby w ciele pętli wartość tej zmiennej zmieniała się, aby doprowadzić do fałszywości Warunku i zakończyć działanie pętli.

Przykład użycia instrukcji Do While:

intLiczba=0 Do While (intLiczba<100) intLiczba=InputBox("Wpisz liczbę całkowitą", "Podaj liczbę") Loop

Pętla typu Do Until jest pętlą podobną do Do While. Jedyna różnica polega na tym, że pętla Do Until wykonuje zawarty w niej blok rozkazów tak długo, jak długo podany warunek jest fałszywy. Wymaga ona zastosowania wyrażenia porównania. Wykorzystuje do tego celu operatory porównania. Pętla ta jest ograniczona słowami kluczowymi Do i Loop. W ciele pętli może znajdować się jedna instrukcja lub cały blok instrukcji VB. Blok ten jest wykonywany tak długo jak długo podany warunek jest fałszywy. Ważne jest więc, aby w ciele pętli umieścić instrukcje, które doprowadzą do tego, że podany warunek kiedyś stanie się prawdziwy. W przeciwnym razie pętla się nie zakończy.

Składnia instrukcji Do Until:

Do Until (Warunek) Blok instrukcji VB wykonywany gdy Warunek jest fałszywy Loop

Zadania

- Napisz program wykonujący jedno z czterech działań na dwóch liczbach. Skorzystaj z InputBox do wyboru działania.
- Napisz program oparty na pętli for wypisujący następujący "tekst":

oooooooooooo
oooooooooooo
oooooooooooo
oooooooooooo
oooooooooooo

Zmodyfikuj program aby wyświetlał:

o
oo
ooo
oooo

- Napisz program wykonujący jedno z czterech działań na dwóch liczbach. Rozbuduj go w taki sposób aby po wykonaniu zadania program prosił o podanie nowego działania. Program ma kończyć działanie po wpisaniu przez użytkownika słowa „koniec”.
- Napisz program wyświetlający liczby pierwsze mniejsze od 100. Czy stosując tablice można przyspieszyć działanie programu?

Obiekty standardowe w WSH

W skład środowiska WSH wchodzi szereg obiektów dostarczających metody, które pozwalają na zarządzanie szeroką gamą elementów systemu operacyjnego. Podstawowym obiektem, który dostarcza najważniejsze metody jest WScript. Metody w nim zawarte pozwalają między innymi na utworzenie instancji pozostałych klas standardowych.

Obiekt WScript

Obiekt WScript jest podstawowym obiektem środowiska WSH. Właściwości WScript przechowują dane o wykonaniu i parametrach uruchomienia skryptu. Metody WScript pozwalają na sterowanie wykonaniem skryptu oraz na zarządzanie obiektami.

Metody WScript:

CreateObject tworzy obiekt określony nazwą.

Składnia:

```
WScript.CreateObject(strProgID[,strPrefix])
```

Echo pozwala na wyświetlenie komunikatu na ekranie (lub konsoli).

Quit służy do kończenia pracy skryptu.

Sleep wstrzymuje skrypt na zadaną liczbę milisekund.

Własności WScript:

Application przechowuje obiekt aplikacji skryptu.

Arguments przechowuje obiekt argumentów (opis poniżej).

FullName przechowuje nazwę (wraz ze ścieżką) interpretera WSH

Name przechowuje nazwę opisową interpretera WSH

Path przechowuje ścieżkę dostępu do interpretera WSH

ScriptFullName przechowuje pełną nazwę (wraz ze ścieżką) wykonywanego skryptu

ScriptName przechowuje nazwę wykonywanego pliku

StdErr przechowuje obiekt strumienia błędów (tylko CScript)

StdIn przechowuje obiekt strumienia wejściowego (tylko CScript)

StdOut przechowuje obiekt strumienia wyjściowego (tylko CScript)

Obiekt WshArguments

Własności WshArguments:

Item zawiera listę parametrów linii poleceń.

Length zawiera liczbę znaków danego parametru.

Count przechowuje liczbę parametrów.

Przykład (należy uruchomić na konsoli przy pomocy CScript i podać jakieś parametry):

```
Set objArgs = WScript.Arguments
For I = 0 to objArgs.Count - 1
    WScript.Echo objArgs(I) Next
```

Obiekt WshEnvironment

Metody WshEnvironment:

Remove Metoda usuwa zmienną środowiskową. Składnia:

```
object.Remove(strName)
```

Własności WshEnvironment:

Item	zawiera listę dostępnych zmiennych.
Length	przechowuje długość danej zmiennej.
Count	przechowuje liczbę dostępnych zmiennych.

Przykład:

```
Set WshShell = WScript.CreateObject("WScript.Shell")
Set WshSysEnv = WshShell.Environment("SYSTEM")
WScript.Echo WshSysEnv("NUMBER_OF_PROCESSORS")
```

Obiekt WshNetwork

Metody WshNetwork:

EnumPrinterConnection zwraca listę dostępnych drukarek.

Składnia:

```
objPrinters = object.EnumPrinterConnections
```

Przykład:

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
Set oPrinters = WshNetwork.EnumPrinterConnections
For i = 0 to oPrinters.Count -1
    WScript.Echo oPrinters.Item(i)
Next
```

AddPrinterConnection Mapuje zdalną drukarkę sieciową. Składnia:

```
object.AddPrinterConnection strLocalName,
strRemoteName[,bUpdateProfile][,strUser][,strPassword]
```

Przykład:

```
Set WshNetwork = CreateObject("WScript.Network")
WshNetwork.AddPrinterConnection "LPT1", "\\Server\Print1"
```

AddWindowsPrinterConnection Mapuje drukarkę sieciową w systemie Windows. Składnia:

```
object.AddWindowsPrinterConnection(strPrinterPath)
```

Przykład:

```
Set WshNetwork = CreateObject("WScript.Network")
PrinterPath = "\\printserv\DefaultPrinter"
PrinterDriver = "Lexmark Optra S 1650"
WshNetwork.AddWindowsPrinterConnection PrinterPath,
PrinterDriver
```

RemovePrinterConnection usuwa mapowanie drukarki.

SetDefaultPrinter

ustawia wybraną drukarkę jako domyślną. Składnia:

```
object.SetDefaultPrinter strPrinterName
```

Przykład:

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
WshNetwork.AddPrinterConnection "LPT1:", "\\Server\Print1"
WshNetwork.SetDefaultPrinter "\\Server\Print1"
```

EnumNetworkDrives - zwraca listę dysków sieciowych. Składnia

```
objDrives = object.EnumNetworkDrive
```

Przykład:

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
Set oDrives = WshNetwork.EnumNetworkDrives
For i = 0 to oDrives.Count -1
    WScript.Echo oDrives.Item(i)
Next
```

MapNetworkDrive mapuje dysk sieciowy. Składnia:

```
object.MapNetworkDrive strLocalName, strRemoteName, [bUpdateProfile], [strUser],
[strPassword]
```

Przykład:

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
WshNetwork.MapNetworkDrive "Z:", "\\Server\Share"
```

RemoveNetworkDrive usuwa dysk sieciowy. Składnia:

```
object.RemoveNetworkDrive strName, [bForce], [bUpdateProfile]
```

Przykład:

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
WshNetwork.MapNetworkDrive "Z:", "\\Server\Share"
WshNetwork.RemoveNetworkDrive "Z:"
```

ComputerName przechowuje nazwę komputera

UserDomain przechowuje bieżącą domenę

UserName przechowuje nazwę użytkownika

Przykład:

```
Set objNet = WScript.CreateObject("WScript.Network")
WScript.Echo objNet.ComputerName & " - " & objNet.UserName
```


Obiekt WshShell

Metody WshShell:

AppActivate aktywuje wybrane okno.

CreateShortcut tworzy skrót. Przykłady:

```
'Utworzenie skrótu do wykonywanego skryptu
Set WshShell = WScript.CreateObject("WScript.Shell")
Set oShellLink = WshShell.CreateShortcut("Current
Script.lnk") oShellLink.TargetPath = WScript.ScriptFullName
oShellLink.Save

'Utworzenie skrótu do strony WWW
Set oUrlLink = WshShell.CreateShortcut("Microsoft Web
Site.URL") oUrlLink.TargetPath = "http://www.microsoft.com/"
oUrlLink.Save
```

LogEvent zapisuje zdarzenie do pliku WSH.log

Popup pokazuje okienko z zadaną treścią. Składnia:

```
IntButton = object.Popup(strText, [natSecondsToWait], strTitle], [natType])
```

Przykład:

```
Set WshShell = WScript.CreateObject("WScript.Shell")
WshShell.Popup "Where do you want to go today?"
```

RegDelete Usunięcie wpisu z rejestru

RegRead Odczytanie wartości z rejestru

RegWrite Wpisanie wartości do rejestru

Przykład:

```
'Utworzenie klucza "Key" i wartości "Value"
Set WshShell = WScript.CreateObject("WScript.Shell")
WshShell.RegWrite "HKCU\ScriptEngine\Value", "Some string value"
WshShell.RegWrite "HKCU\ScriptEngine\Key\", 1 , "REG_DWORD"
'Odczytanie wartości "Value" i wartości domyślnej klucza "Key"
WScript.Echo WshShell.RegRead("HKCU\ScriptEngine\Value")
WScript.Echo WshShell.RegRead("HKCU\ScriptEngine\Key\")
'Usunięcie klucza i wartości
WshShell.RegDelete "HKCU\ScriptEngine\Value"
WshShell.RegDelete
"HKCU\ScriptEngine\Key\"
```

Run pozwala na uruchomienie dowolnej aplikacji. Składnia:

```
object.Run (strCommand, [intWindowStyle], [bWaitOnReturn])
```

Przykład:

```
Set WshShell = WScript.CreateObject("WScript.Shell")
WshShell.Run ("%windir%\notepad " & WScript.ScriptFullName)
```

SendKeys Metod pozwala na symulację naciśnięć klawiszy.

Przykład:

```
set WshShell = CreateObject("WScript.Shell")
WshShell.Run "calc"
WScript.Sleep 100
WshShell.AppActivate "Calculator"
WScript.Sleep 100
WshShell.SendKeys "1{+}"
WScript.Sleep 500
WshShell.SendKeys "2"
WScript.Sleep 500
WshShell.SendKeys "~"
WScript.Sleep 500
WshShell.SendKeys "*3"
WScript.Sleep 500
WshShell.SendKeys "~"
WScript.Sleep 2500
```

Zadania

- Napisz skrypt wyświetlający Architekturę, identyfikator oraz wersję procesora (pobierz informację ze zmiennych systemowych).
- Napisz skrypt startowy dla użytkownika. mapujący jego katalog domowy oraz drukarkę. Dodaj go do ustawień użytkownika.
- Uzupełni poprzedni skrypt w taki sposób aby pokazywał czas poprzedniego logowania (zapamiętaj ten czas w rejestrze).

Wybrane rozszerzenia WSH

Obsługa strumieni

Obiekt WScript posiada własności StdIn, StdOut i StdErr, które reprezentują standardowe strumienie programu konsolowego. Ze względu na charakter tych własności, strumienie przez nie przechowywana, można używać tylko po uruchomieniu skryptu poleceniem 'CScript'.

Strumień StdOut

Strumień StdOut pozwala na wysłanie danych na standardowy strumień wyjściowy (domyślnie konsolę). Posiada następujące metody:

Write	wysyła tekst bez znaku końca linii
WriteLine	wysyła tekst ze znakiem końca linii
WriteBlankLines	wysyła puste linie (znaki końca linii)

Przykład:

```
Set objNetwork = Wscript.CreateObject("Wscript.Network")
Set objStdOut = WScript.StdOut
objStdOut.Write "User: "
objStdOut.Write objNetwork.UserDomain
objStdOut.Write "\"
objStdOut.Write objNetwork.UserName
objStdOut.WriteLine(1)
objStdOut.WriteLine objNetwork.ComputerName
objStdOut.Write "Informacje pobrane."
objStdOut.Close
```

Strumień StdIn

Strumień StdIn pozwala na odczytywanie danych ze standardowego strumienia wejściowego (domyślnie konsoli). Posiada następujące metody:

Read	odczytuje określoną liczbę znaków
ReadLine	odczytuje jedną linię (do wciśnięcia Enter na konsoli – znaku końca linii)
ReadAll	odczytuje cały dostarczony strumień (ma zastosowanie w przypadku danych pochodzących z innego programu)
Skip	omija określoną liczbę znaków
SkipLine	omija określoną liczbę linii
AtEndOfLine	sprawdza czy wystąpił koniec linii
AtEndOfStream	sprawdza czy wystąpił koniec danych/pliku

Przykład:

```
Wscript.StdOut.Write "Podaj liczbę dziesiętną:"
" strDecimal = Wscript.StdIn.ReadLine
Wscript.StdOut.WriteLine strDecimal & " jest równe " & Hex(strDecimal) & "
heksadecymalnie."
```

Istnieje możliwość przekierowania strumieni do programu wywołanego przez skrypt WSH. Użyć należy wówczas obiektów StdIn i StdOut przechowywanych w obiekcie zwracanym przez metodę Exec. Poniżej przykład przekierowania strumienia z programu test.bat:

```

Dim WshShell, oExec, input
Set WshShell = CreateObject("WScript.Shell")
Set oExec = WshShell.Exec("test.bat")
input = ""

Do While True
    If Not oExec.StdOut.AtEndOfStream Then
        input = input & oExec.StdOut.Read(1)
        If InStr(input, "dowolny klawisz . . .") <> 0 Then Exit Do
    End If
    WScript.Sleep 100
Loop
oExec.StdIn.Write " "

Do While oExec.Status <> 1
    WScript.Sleep 100
Loop

```

Plik test.bat dla powyższego przykładu może mieć postać:

```

@echo "Start pliku wsadowego"
@pause

```

Obsługa plików i folderów

WSH dysponuje rozszerzeniem pozwalającym na niemalże nieograniczony dostęp do dysków logicznych, folderów oraz plików. Obiekt, który pozwala na zarządzanie systemem plików to `Scripting.FileSystemObject`.

Najważniejsze obiekty wchodzące w skład `FileSystemObject` to:

Drive

Obiekt reprezentujący dysk logiczny. Przykład zastosowania obiektu tego typu:

```

Dim oFSO
Set oFSO = WScript.CreateObject("Scripting.FileSystemObject")
WScript.Echo oFSO.GetDrive("C:").RootFolder.Path

```

Folder

Obiekt reprezentujący folder. Przykład zastosowania obiektu tego typu:

```

Dim oFSO
Set oFSO = WScript.CreateObject("Scripting.FileSystemObject")
Set oFolder = oFSO.GetFolder("C:\DeleteMe")
oFolder.Delete

```

lub alternatywnie:

```

Dim oFSO
Set oFSO = WScript.CreateObject("Scripting.FileSystemObject")
oFSO.DeleteFolder "C:\DeleteMe"

```

File

Obiekt reprezentujący plik. Przykład zastosowania obiektu tego typu:

```
Dim oFSO
Set oFSO = WScript.CreateObject("Scripting.FileSystemObject")
Set oFile = oFSO.GetFile("C:\MyFile.txt")
oFile.Copy "D:\MyFile.txt"
```

Podobnie jak w przypadku folderu można użyć składni alternatywnej:

```
Dim oFSO
Set oFSO = WScript.CreateObject("Scripting.FileSystemObject")
If oFSO.FileExists("C:\MyFile.txt") Then
    oFSO.CopyFile "C:\MyFile.txt", "D:\MyFile.txt"
End If
```

TextStream

Obiekt pozwalający na zapis do plików tekstowych. Przykład zastosowania:

```
Dim oFSO, oTS
Set oFSO = WScript.CreateObject("Scripting.FileSystemObject")
Set oTS =
oFSO.CreateTextFile("C:\MyOutput.txt")
oTS.WriteLine "Hello, world!" oTS.Close
WScript.Echo "All done!"
```

Inne zastosowania WSH

Możliwości interpretera WSH oraz zastosowania skryptów w Windows są praktycznie nieograniczone. Najlepsze pojęcie o tym co można uzyskać dzięki skryptom daje przeglądanie gotowych skryptów napisanych przez administratorów systemów Windows. Bogata biblioteka skryptów znajduje się na poniższej stronie:

<http://www.sapien.com/forums/viewforum.php?f=19>

Zadania

- Napisz skrypt zapisujący informacje o systemie do pliku tekstowego.
- Napisz skrypt pobierający liczby z pliku i wyświetlający ich sumę (odszukaj funkcję odczytującą z pliku).