

Windows Host Script

WSH udostępnia dwa tryby uruchamiania skryptów: tekstowy i graficzny. O sposobie przetwarzania programu może zadecydować powłoka systemowa

Skrypt utworzony w języku JScript (a dokładniej - w jego odmianie autorstwa Microsoftu) powinien zostać zapisany w pliku z rozszerzeniem JS. Spróbujmy zatem utworzyć najprostszy możliwy program, wypisujący coś na ekranie komputera. Wystarczy do tego jeden wiersz programu:

```
WScript.Echo('Witaj! Jest ' + Date());
```

Po zapisaniu skryptu (np. w pliku skrypt1.js) spróbujmy go uruchomić poprzez dwukrotne kliknięcie ikony tego pliku lewym przyciskiem mysz.

Zapis `WScript.Echo()` oznacza odwołanie się do metody `Echo()` obiektu `WScript`, który jest jednym z najważniejszych elementów modelu obiektowego WSH. Metoda `Echo()` wyświetla na ekranie podany w formie parametru tekst. Przy okazji nauczyliśmy się używać funkcji `Date()`, zwracającej aktualną datę systemową.

Jednym z bardziej przydatnych zastosowań skryptów systemowych (w dowolnym systemie operacyjnym) jest automatyzacja często wykonywanych lub skomplikowanych operacji plikowych. Przykładami mogą być chociażby usuwanie starych plików tymczasowych, tworzenie kopii zapasowych czy kontrolowanie ilości wolnego miejsca na dysku.

Spróbujmy przyjrzeć się mechanizmom obsługi systemu plików przez WSH. Zaczniemy od najprostszych czynności - np. sprawdzenia miejsca zajmowanego przez katalog:

```
var fso;  
fso = new ActiveXObject('Scripting.FileSystemObject');  
var katalog = fso.GetFolder('c:\\Winnt');  
WScript.Echo('Rozmiar folderu wynosi: ' + katalog.Size + ' bajty(ów)');
```

Drugi wiersz skryptu powoduje utworzenie obiektu typu **Scripting.FileSystemObject** i przypisanie zmiennej `fso` wskaźnika do niego. Wspomniany obiekt oferuje metody pozwalające na praktycznie dowolne operacje dyskowe - takie jak tworzenie, usuwanie czy kopiowanie plików i katalogów, operacje na ich atrybutach itd. Zanim jednak zrobimy cokolwiek z plikiem/folderem, trzeba uzyskać wskaźnik do obiektu typu **File** lub **Folder** - dopiero te obiekty udostępniają interesujące nas funkcje. Robimy to w kolejnym wierszu skryptu (`var katalog = fso.GetFolder('C:\\Winnt')`). Podwójny ukośnik "\\ " jest w rzeczywistości interpretowany jako pojedynczy ("\ " to w JScriptcie znak specjalny, po którego wystąpieniu parser oczekuje kolejnego znaku). Zamiast katalogu `C:\Winnt` należy w razie potrzeby podać prawidłowy folder systemowy.

Ostatnie polecenie powoduje wyświetlenie na ekranie tekstu, którego częścią jest wartość zwracana przez metodę `Size()` obiektu `katalog`, czyli całkowity rozmiar folderu (wraz z zawartością). Znaki "+" to w JavaScriptcie operator łączenia (konkatenacji) łańcuchów tekstowych.

Jak wcześniej wspomniano, obiekt **Scripting.FileSystemObject** zapewnia różnorodne operacje plikowe, m.in. z użyciem atrybutów plików/folderów. Możemy np. z łatwością

utworzyć skrypt, wyświetlający listę plików w zadanym katalogu, które spełniają określony warunek. Warunkiem może być np. wskazana data utworzenia. Poniższy skrypt wyświetla pliki w katalogu c:\temp, które charakteryzują się datą modyfikacji późniejszą niż 1 maja 2002 roku:

```
var fso, pliki, data, buf = '';
fso = new ActiveXObject('Scripting.FileSystemObject');
var katalog = FSO.GetFolder('c:\temp');
pliki = new Enumerator(katalog.files);
data = new Date(2002, 5, 1);
for (;!pliki.atEnd(); pliki.moveNext())
if (pliki.item().DateLastModified > data)
buf += pliki.item().Name + ', ' + pliki.item().Size + ' bajty(ów) \n';
WScript.Echo('Pliki o dacie późniejszej niż ' + data.getDate() + '/' + data.getMonth() + '/' +
data.getYear() + ':\n\n' + buf)
```

Obiekt `Scripting.FileSystemObject` i jego metoda `GetFolder()` są nam już znane. Wiersz `pliki = new Enumerator(katalog.files)` powoduje utworzenie kolekcji obiektów reprezentujących pliki, zwróconych przez metodę `files` obiektu `katalog` (typu `Folder`). Zmienna **data** będzie zawierała wartość odpowiadającą 5 maja 2002.

W pętli **for**, powtarzanej aż do napotkania ostatniego elementu kolekcji pliki (sygnalizowanego zwróceniem wartości `true` przez metodę `pliki.atEnd()`), wyszukiwane są obiekty spełniające odpowiedni warunek. Jest nim wyrażenie **`pliki.item().DateLastModified > data`**, którego wartością jest `true`, jeżeli data ostatniej modyfikacji aktualnego pliku jest późniejsza niż ta, którą reprezentuje zawartość zmiennej `data`. Jeśli tak jest, do zmiennej łańcuchowej `buf` dodawany jest wiersz zawierający nazwę i rozmiar znalezionej pliku. Po wykonaniu każdego kroku przesuwany jest wskaźnik aktualnej pozycji listy pliki (**`pliki.moveNext()`**). Ostatnią czynnością wykonywaną przez nasz skrypt jest wyświetlenie listy znalezionych w wskazanym katalogu plików, spełniających podany warunek. Dzieje się to dzięki metodzie `WScript.Echo()`. Metody `getDate()`, `getMonth()` i `getYear()` obiektu `data` zwracają odpowiednio dzień miesiąca, miesiąc i rok odpowiadające dacie przechowywanej w tej zmiennej (gdybyśmy skorzystali bezpośrednio z wartości zmiennej `data`, otrzymalibyśmy datę w domyślnym formacie stosowanym przez WSH).

Modyfikując nieco nasz skrypt, możemy np. uzyskać program usuwający pliki o rozszerzeniu `TMP` z podanego folderu:

```
var fso, pliki, data,
nazwa, buf = '';
fso = new ActiveXObject('Scripting.FileSystemObject');
var katalog = fso.GetFolder('c:\temp');
pliki = new Enumerator(katalog.files);
for (;!pliki.atEnd();
pliki.moveNext())
{
nazwa = fso.GetExtensionName(pliki.item().name);
if (nazwa.toUpperCase() == 'TMP')
fso.DeleteFile(pliki.item());
}
```

Wyjaśnień mogą wymagać jedynie metoda `GetExtensionName()`, zwracająca rozszerzenie podanego pliku, oraz `toUpperCase()`, konwertująca małe litery na wielkie w podanym napisie.

Współpraca z aplikacjami

Ciekawą opcją WSH jest możliwość współpracy z aplikacjami Microsoftu, takimi jak Internet Explorer czy programy wchodzące w skład MS Office'a. Model obiektowy WSH udostępnia odpowiednie klasy, których metody pozwalają na sterowanie pracą aplikacji. W przypadku IE jest to obiekt **Internet-Explorer**, dla Worda utworzono obiekt **Word.Application**, dla Accessa - **Access.Application**. Aby np. wyświetlić okno IE z otwartą stroną autora, wystarczy utworzyć prosty skrypt:

```
var misio = new ActiveXObject ('InternetExplorer.Application');
misio.Navigate('http://klub.chip.pl/pgmys/');
misio.Visible = true;
```

Możemy również "pisać" w pustym oknie przeglądarki - służy do tego celu metoda `write()` obiektu `InternetExplorer.Application.document`:

```
var misio = new ActiveXObject ('InternetExplorer.Application');
misio.Navigate('about:blank');
misio.Toolbar = true;
misio.StatusBar = true;

misio.Resizable = false;

while (misio.Busy) {
    misio.Width = 400;
    misio.Height = 300;
    misio.Left = 30;
    misio.Top = 20;
};
misio.document.write('');
misio.document.write('Przykładowy tekst wywietlony w oknie przeglądarki');
misio.document.write('');
misio.Visible = true;
```

Najpierw musimy otworzyć puste okno przeglądarki (**`misio.Navigate('about: blank')`**). Kolejny krok to wybór żądanych parametrów okna IE (wyświetlane paski, możliwość zmiany rozmiaru okna itp.). Teraz czas na ustalenie rozmiarów okienka (pętla **`while...`**) - zanim to jednak zrobimy, musimy odczekać, aż aplikacja zostanie załadowana do pamięci. Przy okazji wykluczymy sytuację, kiedy program próbuje wyświetlić coś w oknie przeglądarki, zanim system utworzy odpowiednie obiekty.

Gdy przeglądarka jest gotowa do pracy, możemy skorzystać ze wspomnianej metody `write()`. Niestety, musimy podać przeglądarce pełny kod HTML strony (na szczęście nie jest to

trudne). Ostatni krok to właściwe wyświetlenie okna na ekranie (nadanie jego atrybutowi **Visible** wartości **true**).

Przydatne drobiazgi

Wśród wielu możliwości WSH znajdują się i takie, które trudno zakwalifikować do wymienionych kategorii. Należy do nich np. stosowanie argumentów wywołania skryptu, co może się przydać prawie w każdym używanym na co dzień programie. Prześledźmy działanie prostego skryptu, spełniającego zadania podobne do wykonywanych przez DOS-owe polecenie copy:

```
var fso,
    argumenty = WScript.Arguments;
if (argumenty.Count() == 2)
{
fso = new ActiveXObject ('Scripting.FileSystemObject');
if (fso.FileExists(argumenty.item(0)))
{
    fso.CopyFile(argumenty.item(0),
        argumenty.item(1), false)
    }
else
{
    WScript.Echo('Plik do skopiowania nie istnieje');
}
} else
{
    WScript.Echo('Nieprawidłowa liczba argumentów!');
}
```

Z naszego punktu widzenia ważny jest pierwszy wiersz skryptu, a dokładniej przypisanie zmiennej **argumenty** listy parametrów wywołania programu (**argumenty = WScript.Arguments**). Do parametrów odwołujemy się później za pomocą metody **item()** (pierwszy parametr to **argumenty.item(0)**, piąty - **argumenty.item(4)** itd.).

Warto również wspomnieć o zastosowaniach obiektu **Shell**. Pozwala on na korzystanie z elementów interfejsu użytkownika Windows. Możemy np. z łatwością otworzyć dowolny katalog w Eksploratorze czy wyświetlić na ekranie okienko apletu z Panelu sterowania:

```
var szel, folder, plik;
szel = new ActiveXObject('Shell.Application');
szel.Explore('C:\\temp');
szel.ControlPanelItem('desk.cpl');
```

Dostępna jest również opcja uruchomienia z poziomu skryptu dowolnego programu znajdującego się na dysku. Do tego celu służy metoda **Run()** obiektu **Shell**. Jeśli więc chcie-

libyśmy, aby nasz przykładowy skrypt uruchamiał kalkulator Windows, należy do programu dopisać wiersz: `szel.Run('calc.exe')`

Korzystając z powyższego tekstu oraz z informacji zawartych na stronie internetowej <http://republika.pl/winhost/> rozwiąż następujące problemy:

1. Utwórz skrypty zamieszczone w tekście powyżej jako przykłady.
2. Napisz skrypt, który usunie wszystkie pliki ze wskazanego katalogu
3. Napisz skrypt, który otworzy aplikację MS WORD
4. Napisz skrypt, który wyświetli w oknie aktualny czas i datę (w języku polskim)