ОПЕРАЦИОННЫЕ СИСТЕМЫ

ЛАБОРАТОРНАЯ РАБОТА №7

Основы работы с Windows Script Host

Стандартные объекты WSH

С помощью внутренних объектов WSH из сценариев можно выполнять следующие основные задачи:

1) выводить информацию в стандартный выходной поток (на экран) или в диалоговое окно Windows;
2) читать данные из стандартного входного потока (т. е. вводить данные с клавиатуры) или использовать информацию, выводимую другой командой;
3) использовать свойства и методы внешних объектов, а также обрабатывать события, которые генерируются этими объектами;
4) запускать новые независимые процессы или активизировать уже имеющиеся;
5) запускать дочерние процессы с возможностью контроля их состояния и доступа к их стандартным входным и выходным потокам;
6) работать с локальной сетью: определять имя зарегистрировавшегося пользователя, подключать сетевые диски и принтеры;
7) просматривать и изменять переменные среды;
8) получать доступ к специальным папкам Windows;
9) создавать ярлыки Windows;
10) работать с системным реестром.

В WSH входят перечисленные ниже объекты:

1) WScript. Это главный объект WSH, который служит для создания других объектов или связи с ними, содержит сведения о сервере сценариев, а также позволяет вводить данные с клавиатуры и выводить информацию на экран или в окно Windows.
2) WshArguments. Обеспечивает доступ ко всем параметрам командной строки запущенного сценария или ярлыка Windows.
3) WshNamed. Обеспечивает доступ к именным параметрам командной строки запущенного сценария.
4) WshUnnamed. Обеспечивает доступ к безымянным параметрам командной строки запущенного сценария.
5) WshShell. Позволяет запускать независимые процессы, создавать ярлыки, работать с переменными среды, системным реестром и специальными папками Windows.
6) WshSpecialFolders. Обеспечивает доступ к специальным папкам Windows.
7) WshShortcut. Позволяет работать с ярлыками Windows.
8) WshUrlShortcut. Предназначен для работы с ярлыками сетевых ресурсов.
9) WshEnvironment. Предназначен для просмотра, изменения и удаления переменных среды.
10) WshNetwork. Используется при работе с локальной сетью: содержит сетевую инфомацию для локального компьютера, позволяет подключать сетевые диски и принтеры.
11) WshScriptExec. Позволяет запускать консольные приложения в качестве дочерних процессов, обеспечивает контроль состояния этих приложений и доступ к их стандартным входным и выходным потокам.
12) WshController. Позволяет запускать сценарии на удаленных машинах.
13) WshRemote. Позволяет управлять сценарием, запущенным на удаленной машине.
14) WshRemoteError. Используется для получения информации об ошибке, возникшей в результате выполнения сценария, запущенного на удаленной машине.

Кроме этого, имеется объект *FileSystemObject*, обеспечивающий доступ к файловой системе компьютера.

Объект WScript

Свойства объекта *WScript* позволяют получить полный путь к использующемуся серверу сценариев (*wscript.exe* или *cscript.exe*), параметры командной строки, с которыми запущен сценарий, режим его работы (интерактивный или пакетный). Кроме этого, с помощью свойств объекта *WScript* можно выводить информацию в стандартный выходной поток и читать данные из стандартного входного потока. Также *WScript* предоставляет методы для работы внутри сценария с объектами автоматизации и вывода информации на экран (в текстовом режиме) или в окно Windows. В сценарии WSH объект *WScript* можно использовать сразу, без какого-либо предварительного описания или создания, т. к. его экземпляр создается сервером сценариев автоматически. Для использования же всех остальных объектов нужно применять либо метод *CreateObject*, либо определенное свойство другого объекта.

Таблица. Свойства объекта *WScript*

Свойство	**Описание**
Application	Предоставляет интерфейс ***I****dispatch* для объекта *WScript*
Arguments	Содержит указатель на коллекцию *WshArguments*, в которой находятся параметры командной строки для исполняемого сценария
FullName	Содержит полный путь к исполняемому файлу сервера сценариев (обычно это C:\W1ND0WS\SYSTEM32\CSCRIPT.EXE или C:\WINDOWS\SYSTEM32\WSCRIPT.EXE)
Name	Содержит название объекта *WScript* (Windows Script Host)
Path	Содержит путь к каталогу, в котором находится *cscript.exe* или *wscript.exe* (обычно это C:\WINDOWS\SYSTEM32)
ScriptFullName	Содержит полный путь к запущенному сценарию
ScriptName	Содержит имя запущенного сценария
StdErr	Позволяет запущенному сценарию записывать сообщения в стандартный поток для ошибок
StdIn	Позволяет запущенному сценарию читать информацию из стандартного входного потока
StdOut	Позволяет запущенному сценарию записывать информацию в стандартный выходной поток
Version	Содержит версию WSH

Свойство Arguments

В следующем примере с помощью цикла *for* на экран выводятся все параметры командной строки, с которыми был запущен сценарий.

```
/*Пример 2.1. Вывод на экран всех параметров сценария */
/*********************************************/
/* Имя: ShowArgs.js */
/* Язык: JScript */
/* Описание: Вывод на экран параметров запущенного сценария */
var i, objArgs;
objArgs - WScript.Arguments; //Создаем объект WshArguments
for (i=0; i<=objArgs.Count()-l; i++)
WScript.Echo(objArgs(i)); //Выводим на экран i-й аргумент
/************ Конец ************************************/
```

Свойства StdErr, StdIn, StdOut

Доступ к стандартным входным и выходным потокам с помощью свойств *StdIn, StdOut* и *StdErr* можно получить только в том случае, если сценарий запускался в консольном режиме с помощью *cscript.exe*. Если сценарий был запущен с помощью *wscript.exe*, то при попытке обратиться к этим свойствам возникнет ошибка "invalid Handle".

Работать с потоками *StdOut* и *StdErr* можно с помощью методов *Write, WriteLine, WriteBlankLines*, а с потоком *StdIn* — с помощью методов *Read, ReadLine, ReadAll, Skip, SkipLine*. Эти методы кратко описаны в таблице.

Таблица. Методы для работы с потоками

Свойство	**Описание**
Read(n)	Считывает из потока *StdIn* заданное параметром *n* число символов и возвращает полученную строку
ReadAll()	Читает символы из потока *StdIn* до тех пор, пока не встретится символ конца файла ASCII 26 (<Ctrl>+<Z>), и возвращает полученную строку
ReadLine()	Возвращает строку, считанную из потока *StdIn*
Skip(n)	Пропускает при чтении из потока *StdIn* заданное параметром *n* число символов
SkipLine()	Пропускает целую строку при чтении из потока *StdIn*
Write(*string*)	Записывает в поток *StdOut* или *StdErr* строку *string* (без символа конца строки)
WriteBlankLines(n)	Записывает в поток *StdOut* или *StdErr* заданное параметром *n* число пустых строк
WriteLine(string)	Записывает в поток *StdOut* или *StdErr* строку *string*(вместе с символом конца строки)

Операционная система Windows поддерживает механизм конвейеризации (символ “|” в командной строке). Этот механизм делает возможным передачу данных от одной программы к другой. Таким образом, используя стандартные входные и выходные потоки, можно из сценария обрабатывать строки вывода другого приложения или перенаправлять выводимые сценарием данные на вход программ-фильтров (find или sort). Например, следующая команда будет сортировать строки вывода сценария *example.js* и выводить их в файл *sort.txt*:

cscript //Nologo example.js | sort > sort.txt

Опция *//Nologo* здесь нужна для того, чтобы в файл *sort.txt* не попадали строки с информацией о разработчике и номере версии WSH. Кроме этого, с помощью методов, работающих с входным потоком *StdIn*, можно организовывать диалог с пользователем, т. е. создавать интерактивные сценарии. Пример такого сценария представлен ниже.

```
/*Пример 2.2. Пример интерактивного сценария*/
/*************************************************************/
/* Имя:- Interact.js */
/* Язык: JScript */
/* Описание: Ввод/вывод строк в консольном режиме */
/*************************************************************/
var s;
//Выводим строку на экран
WScript.StdOut.Write("Введите число: ");
//Считываем строку
s =WScript.Stdln.ReadLine();
//Выводим строку на экран
WScript.StdOut.WriteLine("Вы ввели число " + s);
/************ конец **********************************/
```

Объект *WScript* имеет несколько методов, которые описаны в таблице.

Таблица. Методы объекта *WScript*

Свойство	**Описание**
CreateObject(*strProgID [,strPrefix])*	Создает объект, заданный параметром *strProgID*
ConnectObject(strObject, strPrefix)	Устанавливает соединение с объектом *strObject*, позволяющее писать функции-обработчики его событий (имена этих функций должны начинаться с префикса *strPrefix*).
*DisconnectObject(*obj)	Отсоединяет объект *obj*, связь с которым была предварительно установлена в сценарии
Echo([Arg1] [, Arg2] [,…])	Выводит текстовую информацию на консоль или в диалоговое окно
GetObject(strPathname [,strProgID], [strPrefix])	Активизирует объект автоматизации, определяемый заданным файлом (параметр *strPathName*), или объект, заданный параметром *strProgID*
Quit([intErrorCode])	Прерывает выполнение сценария с заданным параметром *intErrorCode* кодом выхода. Если параметр *intErrorCode* не задан, то объект *WScript* установит код выхода равным нулю
Sleep(intTime)	Приостанавливает выполнения сценария (переводит его в неактивное состояние) на заданное параметром *intTime* число миллисекунд

Метод CreateObject

Строковый параметр *strProgID*, указываемый в методе *CreateObject*, называется программным идентификатором объекта (Programmic Identifier, ProgID). Если указан необязательный параметр *strPrefix*, то после создания объекта в сценарии можно обрабатывать события, возникающие в этом объекте (естественно, если объект предоставляет интерфейсы для связи с этими событиями). Когда объект сообщает о возникновении определенного события, сервер сценариев вызывает функцию, имя которой состоит из префикса *strPrefix* и имени этого события. Например, если в качестве *strPrefix* указано "*MYOBJ_*", а объект сообщает о возникновении события "*onBegin*", то будет запущена функция "*MYOBJ_onBegin*", которая должна быть описана в сценарии. В следующем примере метод *CreateObject* используется для создания объекта *WshNetwork*:

var WshNetwork = WScript.CreateObject("WScript.Network");

Отметим, что объекты автоматизации из сценариев можно создавать и без помощи WSH. В JScript для этого используется объект ActiveXObject, например:

var WshNetwork = new ActiveXObject("WScript.Network");

В VBscript для создания объектов может использоваться специальная функция *CreateObject*, например:

Set WshNetwork = CreateObject("WScript.Network")

Однако организовать в сценарии обработку событий создаваемого объекта можно только при использовании метода *WScript.CreateObject*.

Метод ConnectObject

Объект, соединение с которым осуществляется с помощью метода *ConnectObject*, должен предоставлять интерфейс к своим событиям. В следующем примере в переменной *MyObject* создается абстрактный объект "*SomeObject*", затем из сценария вызывается метод *SomeMetod* этого объекта. После этого устанавливается связь с переменной *MyObject* и задается префикс "*MyEvent*" для процедур обработки события этого объекта. Если в объекте возникнет событие с именем "*Event*", то будет вызвана функция *MyEvent_Event*. Метод *DisconnectObject* объекта *WScript* производит отсоединение объекта *MyObject*.

```
var MyObject = WScript.CreateObject("SomeObject");
MyObject.SomeMethod();
WScript.ConnectObject(MyObject,"MyEvent");
function MyEvent_Event(strName){
WScript.Echo(strName);
}
WScript.DisconnectObject(MyObject);
```

Метод DisconnectObject

Если соединения с объектом *obj* не было установлено, то метод *DisconnectObject(obj)* не будет производить никаких действий. Пример применения *DisconnectObject* был приведен выше.

Метод Echo

Параметры *Arg1, Arg2* задают аргументы для вывода. Если сценарий был запущен с помощью *wscript.exe*, то метод *Echo* направляет вывод в диалоговое окно, если же для выполнения сценария применяется *cscript.exe*, то вывод будет направлен на экран (консоль). Каждый из аргументов при выводе будет разделен пробелом. В случае использования *cscript.exe* вывод всех аргументов будет завершен символом новой строки. Если в методе *Echo* не задан ни один аргумент, то будет напечатана пустая строка. Например, после выполнения сценария *EchoExample.js* (пример 2.3) с помощью *cscript.exe* на экран будут выведены пустая строка, три числа и строка текста.

```
/*Пример 2.3. Сценарий EchoExample.js*/
/*************************************************************/
/* Имя: EchoExatnple. js */
/* Язык: JScript */
/* Описание: Использование метода WScript.Echo */
/************************************************************ /
WScript.Echo(); //Выводим пустую строку
WScript.Echo(1,2,3); //Выводим числа
WScript.Echo("Привет!"); //Выводим строку
/************ Конец ************************************/
```

Метод Sleep

В следующем примере сценарий переводится в неактивное состояние на 5 секунд:

```
WScript.Echo("Сценарий запущен, отдыхаем...");
WScript.Sleep(5000);
WScript.Echo("Выполнение завершено");
```

Метод *Sleep* необходимо применять при асинхронной работе сценария и какой-либо другой задачи, например, при имитации нажатий клавиш в активном окне с помощью метода WshShell.SendKeys.

Объекты-коллекции

В WSH входят объекты, с помощью которых можно получить доступ к коллекциям, содержащим следующие элементы:

1) параметры командной строки запущенного сценария или ярлыка Windows (объекты WshArguments, WshNamed и WshUnnamed);
2) значения переменных среды (объект WshEnvironment);
3) пути к специальным папкам Windows (объект WshSpeciaiFolders).

Объект WshArguments

Объект *WshArguments* содержит коллекцию всех параметров командной строки запущенного сценария или ярлыка Windows. Этот объект можно создать только с помощью

свойства *Arguments* объектов *WScript* и *WshShortcut*. В принципе, работать с элементами коллекции *WshArguments* можно стандартным для JScript образом — создать объект *Enumerator* и использовать его методы *MoveNext, Item* и *AtEnd*. Например, вывести на экран все параметры командной строки, с которыми запущен сценарий, можно следующим образом.

```
/*Пример 2.4. Вывод всех параметров сценария */
/*********************************************************/
/* Имя: EnumArgs.js */
/* Язык: JScript */
/* Описание: Вывод на экран параметров запущенного сценария */
/*********************************************************/
var objArgs, e, x;
objArgs = WScript.Arguments; //Создаем объект WshArguments
//Создаем объект Enumerator для коллекции objArgs
e = new Enumerator(objArgs);
for (;!e.atEnd();e.moveNext()) {
x = e.item(); //Получаем значение элемента коллекции
WScript.Echo(x); //Выводим значение параметра на экран
}
/************ Конец ***********************************/
```

Однако намного удобнее использовать методы *Count* и *Item* самого объекта *WshArguments* (метод *Item* имеется у всех коллекций WSH). Метод *Count* возвращает число элементов в коллекции, т. е. количество аргументов командной строки, а метод *Item(n)* — значение n-го элемента коллекции (нумерация начинается с нуля). Более того, чтобы получить значение отдельного элемента коллекции *WshArguments*, можно просто указать его индекс в круглых скобках после имени объекта. Число элементов в коллекции хранится и в свойстве *Length* объекта *WshArguments*. Таким образом, предыдущий пример можно переписать более компактным образом.

```
/*Пример 2.5. Вывод всех параметров сценария (методы WSH) */
/*****************************************************/
/* Имя: ShowArgs.js */
/* Язык: JScript */
/* Описание: Вывод на экран параметров запущенного сценария */
/*****************************************************/
var i, objArgs;
objArgs = WScript.Arguments; //Создаем объект WshArguments
for (i-0; i<=objArgs. Count 0-1;
WScript.Echo(objArgs(i)); //Выводим значение i-го параметра
/************ Конец ***********************************/
```

С помощью объекта WshArguments можно также выделять и отдельно обрабатывать аргументы сценария, у которых имеются имена (например, /NametAndrey) и безымянные аргументы. Ясно, что использование именных параметров более удобно, т. к. в этом случае нет необходимости запоминать, в каком порядке должны быть записаны параметры при запуске того или иного сценария. Для доступа к именным и безымянным аргументам используются соответственно два специальных свойства объекта WshArguments: Named и Unnamed. Свойство Named содержит ссылку на коллекцию WshNamed, свойство Unnamed — на коллекцию WshUnnamed. Таким образом, обрабатывать параметры командной строки запущенного сценария можно тремя способами:

1) просматривать полный набор всех параметров (как именных, так и безымянных) с помощью коллекции WshArguments;

2) выделить только те параметры, у которых есть имена (именные параметры) с помощью коллекции WshNamed;

3) выделить только те параметры, у которых нет имен (безымянные параметры) с помощью коллекции WshUnnamed.

У объекта WshArguments имеется еще один метод - ShowUsage. Этот метод служит для вывода на экран информации о запущенном сценарии (описание аргументов командной строки, пример запуска сценария и т.д.). В свою очередь, подобную информацию можно задать только при использовании WSH-сценариев с разметкой XML.

Объект WshEnvironment

Объект WshEnvironment позволяет получить доступ к коллекции, содержащей переменные среды заданного типа (переменные среды операционной системы, переменные среды пользователя или переменные среды текущего командного окна). Этот объект можно создать с помощью свойства Environment объекта WshShell или одноименного его метода:

```
var WshShell=WScript.CreateObject("WScript.Shell"),
WshSysEnv=WshShell.Environment,
WshUserEnv=WshShell.Environment("User");
```

Объект WshEnvironment имеет свойство Length, в котором хранится число элементов в коллекции (количество переменных среды), и методы Count и Item. Для того чтобы получить значение определенной переменной среды, в качестве аргумента метода Item указывается имя этой переменной в двойных кавычках. В следующем примере мы выводим на экран значение переменной среды Path:

```
var WshShell=WScript.CreateObject("WScript.Shell"),
WshSysEnv=WshShell.Environment;
WScript.Echo("Системный путь:", WshSysEnv.Item("Path"));
```

Можно также просто указать имя переменной в круглых скобках после имени объекта:

```
WScript.Echo("Системный путь:",WshSysEnv("Path");
```

Кроме этого, у объекта WshEnvironment имеется метод Remove(strName), который удаляет заданную переменную среды. Например, в примере 2.6 приведен сценарий, который удаляет две переменные (example_1 и example_2) из окружения среды пользователя. Если в окружении среды пользователя нет переменных с именами example_1 и example_2, то при вызове метода Remove произойдет ошибка.

```
/*Пример 2.6. Удаление переменных среды */
/*************************************************************/
/* Имя: RemEnv.js */
/* Язык: JScript */
/* Описание: Удаление двух переменных среды */
/*************************************************************/
//Создаем объект WshShell
var WshShell = WScript.CreateObject("WScript.Shell");
//Создаем объект WshEnvironment
var WshUsrEnv = WshShell.Environment("User");
//Удаляем переменные среды
WshUsrEnv.Remove("EXAMPLE_1");
WshUsrEnv.Remove("EXAMPLE_2");
/************* Конец ***************************************/
```

Объект WshSpecialFolders

Объект WshSpecialFolders обеспечивает доступ к коллекции, содержащей пути к специальным папкам Windows (например, к рабочему столу или к меню Пуск (Start)); задание путей к таким папкам может быть необходимо, например, для создания непосредственно из сценария ярлыков на рабочем столе.

Поддерживаются следующие имена специальных папок:

Desktop; Programs; Favorites; Recent; Fonts; SendTo; MyDocuments; StartMenu; NetHood; Startup; PrintHood; Templates; AllUsersDesktop; AllUsersPrograms; AllUsersStartMenu; AllUsersStartup.

Объект WshSpecialFolders создается с помощью свойства SpeciaiFolders объекта WshShell:
var WshShell=WScript.CreateObject("WScript.Shell"),
WshSpecFold-WshShell.SpeciaiFolders;
Как и почти все коллекции WSH, объект WshSpecialFolders имеет свойство Length и методы Count и Item. Доступ к отдельному элементу производится либо через имя соответствующей папки, либо через числовой индекс (пример 2.7).

```
/*Пример 2.7. Обработка коллекции WshSpecialFolders*/
/*********************************************************/
/* Имя: ShowSpecFold.js */
/* Язык: JScript */
/* Описание: Вывод на экран названий специальных папок Windows*/
/* (коллекция WshSpecialFolders) */
/*********************************************************/
var WshShell, WshFldrs, i;
//Создаем объект WshShell
WshShell = WScript.CreateObj ect("Wscript.Shell");
//Создаем объект WshSpecialFolders
WshFldrs = WshShell.SpeciaiFolders;
WScript.Echo("Некоторые специальные папки...");
//Выводим путь к папке Desktop
WScript.Echo("Desktop="+ WshFldrs.item ("Desktop"));
//Выводим путь к папке Favorities
WScript.Echo("Favorites="+ WshFldrs("Favorites"));
//Выводим путь к папке Programs
WScript.Echo("Programs="+ WshFldrs("Programs"));
WScript.Echo("") ;
WScript.Echo("Список всех специальных папок...");
for (i-0;i<= WshFldrs.Count()-1;i++){
//Выводим на экран i-й элемент коллекции WshFldrs
WScript.Echo(WshFldrs(i)); }
/************ Конец ************************************/
```

Объект WshShell

С помощью объекта WshShell можно запускать новый процесс, создавать ярлыки, работать с системным реестром, получать доступ к переменным среды и специальным папкам Windows. Создается этот объект следующим образом:
var WshShell=WScript.CreateObject("WScript.Shell") ;
Объект WshShell имеет три свойства, которые приведены в таблице.

Таблица. Свойства объекта WshShell

Свойство	**Описание**
CurrentDirectory	Здесь хранится полный путь к текущему каталогу (к каталогу, из которого был запущен сценарий)
Environment	Содержит объект WshEnvironment, который обеспечивает доступ к переменным среды операционной системы.
SpecialFolders	Содержит объект WshSpecialFolders для доступа к специальным папкам Windows (рабочий стол, меню Пуск (Start) и т. д.)

Таблица. Методы объекта WshShell

Свойство	**Описание**
AppActivate(titie)	Активизирует заданное параметром title окно приложения. Строка title задает название окна

	(например, "calc" или "notepad") или идентификатор процесса (Process ID, PID)
CreateShortcut(strPathname)	Создает объект WshShortcut для связи с ярлыком Windows (расширение lnk) или объект WshUrlShortcut для связи с сетевым ярлыком (расширение .url). Параметр strPathname задает полный путь к создаваемому или изменяемому ярлыку
Environment(strType)	Возвращает объект WshEnvironment, содержащий переменные среды заданного вида
Exec(strCommand)	Создает новый дочерний процесс, который запускает консольное приложение, заданное параметром strCommand. В результате возвращается объект WshScriptExec, позволяющий контролировать ход выполнения запущенного приложения и обеспечивающий доступ к потокам StdIn, StdOut и StdErr этого приложения
ExpandEnvironmentStrings(strString)	Возвращает значение переменной среды текущего командного окна, заданной строкой strString (имя переменной должно быть окружено знаками"%")
LogEvent (int Type, strMessage [,strTarget])	Протоколирует события в журнале Windows или в файле WSH.log. Целочисленный параметр intType определяет тип сообщения, строка strMessage – текст сообщения. Метод LogEvent возвращает true, если событие записано успешно и false в противном случае.
Popup(strText, [nSecToWait], [strTitle], [nType])	Выводит на экран информационное окно с сообщением, заданным параметром strText. Параметр nSecToWait задает количество секунд, по истечении которых окно будет автоматически закрыто, параметр strTitle определяет заголовок окна, параметр nType указывает тип кнопок и значка для окна.
RegDelete(strName)	Удаляет из системного реестра заданный параметр или раздел целиком
RegRead(strName)	Возвращает значение параметра реестра или значение по умолчанию для раздела реестра
RegWrite(strName, anyValve [,strType])	Записывает в реестр значение заданного параметра или значение по умолчанию для раздела
Run[strCommand, [intWindowStyle], [bWaitOnReturn])	Создает новый независимый процесс, который запускает приложение, заданное параметром strCommand
SendKeys(string)	Посылает одно или несколько нажатий клавиш в активное окно (эффект тот же, как если бы вы нажимали эти клавиши на клавиатуре)
SpecialFolders(strSpecFolder)	Возвращает строку, содержащую путь к специальной папке Windows, заданной параметром strSpecFolder

Метод Popup

Если в методе не задан параметр strTitle, то по умолчанию заголовком окна будет "Windows Script Host." Параметр nType может принимать те же значения, что и в функции MessageBox

из Microsoft Win32 API. В таблице описаны некоторые возможные значения параметра nType и их смысл (полный список значений этого параметра можно посмотреть в описании функции MessageBox в документации по функциям Windows API).

Таблица. Типы кнопок и иконок для метода Popup

Значение nType	**Константа Visual Basic**	**Описание**
0	vbOkOnly	Выводится кнопка OK
1	vbOkCancel	Выводятся кнопки OK и Отмена (Cancel)
2	vbAbortRetrylgnore	Выводятся кнопки Стоп (Abort), Повтор (Retry) и Пропустить (Ignore)
3	vbYesNoCancel	Выводятся кнопки Да (Yes), Нет (No) и Отмена (Cancel)
4	vbYesNo	Выводятся кнопки Да (Yes) и Нет (No)
5	vbRetryCancel	Выводятся кнопки Повтор (Retry) и Отмена (Cancel)
16	vbCritical	Выводится значок Stop Mark
32	vbQuestion	Выводится значок Question Mark
48	vbExclamation	Выводится значок Exclamation Mark
64	vblnformation	Выводится значок Information Mark

В сценариях, написанных на языке VBScript, можно непосредственно использовать именованные константы типа vbOkCancel без предварительного их объявления. Для того чтобы использовать такие константы в JScript-сценариях, их нужно предварительно объявить, как переменные и присвоить нужные значения (например, var vbOkCancel1;). Естественно, в любых сценариях вместо имен констант можно использовать их числовые значения.

В методе Popup можно комбинировать значения параметра, приведенные в таблице. Например, в результате выполнения следующего сценария:

var WshShell - WScript.CreateObject("WScript.Shell");
WshShell.Popup("Копирование завершено успешно",5,"Ура", 65);

На экран будет выведено информационное окно, которое автоматически закроется через 5 секунд.

Метод Popup возвращает целое значение, с помощью которого можно узнать, какая именно кнопка была нажата для выхода.

Таблица. Возвращаемые методом Popup значения

Значение	**Константа Visual Basic**	**Описание**
-1		Пользователь не нажал ни на одну из кнопок в течение времени, заданного параметром nSecToWait
1	vbOk	Нажата кнопка OK
2	vbCance	Нажата кнопка Отмена (Cancel)
3	vbAbort	Нажата кнопка Стоп (Abort)
4	vbRetry	Нажата кнопка Повтор (Retry)
5	vblgnore	Нажата кнопка Пропустить (Ignore)
6	vbYes	Нажата кнопка Да (Yes)
7	vbN	Нажата кнопка Нет (No)

Объект WshShortcut

С помощью объекта WshShortcut можно создать новый ярлык Windows или изменить свойства уже существующего ярлыка. Этот объект можно создать только с помощью

метода CreateShortcut объекта WshShell. Ниже представлен пример сценария, в котором создается ярлык на этот самый сценарий (ярлык будет находиться в текущем каталоге).

```
/*Пример 2.8 .Создание ярлыка на выполняемый сценарий*/
47
/* Имя: MakeShortcutl.js */
/* Язык: JScript */
/* Описание: Создание ярлыка на выполняемый сценарий */
/*****************************************************/
var WshShell,oShellLink; //Создаем объект WshShell
WshShell = WScript.CreateObject("WScript.Shell");
//Создаем ярлык в текущем каталоге
oShellLink = WshShell.CreateShortcut("Current Script.lnk");
//Устанавливаем путь к файлу
oShellLink.TargetPath = WScript.Script FullName;
//Сохраняем ярлык
oShellLink.Save();
/**********Конец ************************************/
```

Свойства объекта WshShortcut описаны в таблице.

Таблица. Свойства объекта WshShortcut

Свойство	**Описание**
Arguments	Содержит строку, задающую параметры командной строки для ярлыка
Description	Содержит описание ярлыка
FullName	Содержит строку с полным путем к ярлыку
HotKey	Задает "горячую" клавишу для ярлыка, т. е. определяет комбинацию клавиш, с помощью которой можно запустить или сделать активной программу, на которую указывает заданный ярлык
IconLocation	Задает путь к значку ярлыка
TargetPath	Устанавливает путь к файлу, на который указывает ярлык
WindowStyle	Определяет вид окна для приложения, на которое указывает ярлык
WorkingDirectory	Задает рабочий каталог для приложения, на которое указывает ярлык

Свойство Arguments

В листинге приведен пример сценария, создающего ярлык на этот самый сценарий с двумя параметрами командной строки.

```
/*Пример 2.8. Создание ярлык на ценарий с двумя параметрами
командной строки*/
/*********************************************************/
/* имя: MakeShortcut2.js */
/* Язык: JScript */
48
/* Описание: Создание ярлыка на выполняемый сценарий с */
/* аргументами командной строки */
/*********************************************************/
var WshShell, oShellLink;
//Создаем объект WshShell
WshShell = WScript.CreateObject("WScript.Shell");
//Создаем ярлык в текущем каталоге
oShellLink = WshShell.CreateShortcut("Current Script.lnk");
//Устанавливаем путь к файлу
oShellLink.TargetPath = WScript.ScriptFullName;
```

```
//Указываем аргументы командной строки
oShellLink.Arguments = "-a abc.txt";
//Сохраняем ярлык
oShellLink.Save();
/************ Конец ************************************/
```

Свойство HotKey

Для того чтобы назначить ярлыку "горячую" клавишу, необходимо в свойство HotKey записать строку, содержащую названия нужных клавиш, разделенные символом "+". "Горячие" клавиши могут быть назначены только ярлыкам, которые расположены на рабочем столе Windows или в меню Пуск (Start). Для того чтобы нажатия "горячих" клавиш срабатывали, необходимо, чтобы языком по умолчанию в операционной системе был назначен английский. В следующем примере на рабочем столе создается ярлык для Блокнота, которому назначается комбинация "горячих" клавиш <Ctrl>++<Alt>+<D>.

```
/*Пример 2.9. Назначение горячих клавиш на ярлык*/
/*****************************************************/
/* Имя: MakeShortcut3,js */
/* Язык: JScript */
/* Описание: Создание ярлыка на Блокнот с комбинацией "горячих" */
/* клавиш */
/****** ************************************************/
var WshShell,strDesktop,oMyShortcut;
//Создаем объект WshShell
WshShell = WScript.CreateObject("WScript.Shell");
//Определяем путь к рабочему столу
strDesktop = WshShell.SpecialFolders("Desktop");
//Создаем ярлык в текущем каталоге
oMyShortcut = WshShell.CreateShortcut(strDesktop+"\\a_key.lnk");
49
//Устанавливаем путь к файлу
oMyShortcut.TargetPath =WshShell.ExpandEnvironmentStrings
("%windir%\\notepad.exe");
//Назначаем комбинацию "горячих" клавиш
oMyShortcut.Hotkey = "CTRL+ALT+D";
//Сохраняем ярлык oMyShortcut.Save();
WScript.Echo("Горячие" клавиши для ярлыка: "+oMyShortcut.Hotkey);
/*********** Конец ***************************************/
```

Свойство IconLocation

Для того чтобы задать значок для ярлыка, необходимо в свойство IconLocation записать строку следующего формата: "путь, индекс". Здесь параметр путь определяет расположение файла, содержащего нужный значок, а параметр индекс — номер этого значка в файле (номера начинаются с нуля). В следующем примере создается ярлык на выполняющийся сценарий с первым значком (индекс 0) из файла notepad.exe.

```
Пример 2.10.
/*****************************************************/
/* Имя: MakeShortcut4.js */
/* Язык: JScript */
/* Описание: Создание ярлыка на выполняемый сценарий со */
/*значком из notepad.exe */
/*****************************************************/
var WshShell,oShellLink;
//Создаем объект WshShell
WshShell = WScript.CreateObject("WScript.Shell");
```

```
//Создаем ярлык в текущем каталоге
oShellLink = WshShell.CreateShortcut("Current Script.Ink");
//Устанавливаем путь к файлу
oShellLink.TargetPath = WScript.ScriptFullName;
//Выбираем значок из файла notepad.exe
oShellLink.IconLocation = "notepad.exe, 0";
//Сохраняем ярлык oShellLink.Save();
/************ Конец ************************************** /
```

Свойство WindowStyle

Значением свойства WindowStyle является целое число IntWindowStyie которое может принимать значения, приведенные в таблице.

Таблица. Значения параметра IntWindowStyie

IntWindowStyle	Описание
1	Стандартный размер окна. Если окно было минимизировано или максимизировано, то будут восстановлены его первоначальные размеры и расположение на экране
3	Окно при запуске приложения будет развернуто на весь экран (максимизировано)
7	Окно при запуске приложения будет свернуто в значок (минимизировано)

Свойство WorkingDirectory

В следующем примере создается ярлык для Блокнота, причем в качестве рабочего каталога указан корневой каталог диска C:.

Пример 2.11.

```
/*********************************************************/
/* Имя: MakeShortcutS.js */
/* Язык: JScript */
/* Описание: Создание ярлыка на Блокнот с изменением рабочего */
/* каталога */
/*********************************************************/
var WshShell,oShellLink;
//Создаем объект WshShell
WshShell = WScript.CreateObject("WScript.Shell");
//Создаем ярлык в текущем каталоге
oShellLink = WshShell.CreateShortcut("Notepad.Ink");
//Устанавливаем путь к файлу
oShellLink.TargetPath = "notepad.exe";
//Назначаем рабочий каталог
oShellLink.WorkingDirectory = "C:\\";
//Сохраняем ярлык
oShellLink.Save();
/************ Конец**************************************/
```

Объект WshShortcut имеет единственный метод Save, который сохраняет заданный ярлык в каталоге, указанном в свойстве FullName.

Задания к лабораторной работе

Задание 1. Работа с параметрами командной строки

Создайте сценарий JScript, который в зависимости от введенного пользователем параметра командной строки выполняет следующие действия:

а) Сравнение двух чисел, введенных с клавиатуры
б) Выводит на экран путь к исполняемому файлу сервера сценариев, имя запущенного сценария и версию WSH
Предусмотрите вывод справки по работе скрипта.

Задание 2. Работа с объектами коллекциями

Создайте сценарий JScript, который в зависимости от введенного пользователем параметра командной строки выполняет следующие действия:
а) Выводит имена и содержимое всех системных переменных окружения
б) Выводит все специальные папки.
Если введен неверный аргумент командной строки, необходимо предусмотреть сообщение об ошибке.

Задание 3. Работа с методом Popup и WshShortcut

Создайте сценарий JScript, который выполняет следующие действия:
а) При помощи всплывающего окна запрашивает у пользователя необходимость создания на рабочем столе ярлыков для запуска сценария из задания 2.
б) При положительном ответе пользователя создает два ярлыка для выполнения задания 2.а и 2.б
Предусмотрите вывод всех возможных ошибок в журнал.