ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
"МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ им. Н.П. ОГАРЕВА"

А.В. ПОПОВ

СЦЕНАРИИ WINDOWS SCRIPT HOST

УЧЕБНОЕ ПОСОБИЕ

САРАНСК
ИЗДАТЕЛЬСТВО МОРДОВСКОГО УНИВЕРСИТЕТА
2009

УДК 004.384 (075.8)
ББК 397
П58

Рецензенты:
кафедра информационно-вычислительных систем Саранского кооперативного института (филиал) АНО ВПО ЦС РФ "Российский университет кооперации";
директор ИМЦ МРИО, заслуженный работник образования Республики Мордовия
Т.П. Лунина

П58 Сценарии Windows Script Host : учеб. пособие / А.В. Попов. – Саранск : Изд-тво Мордов. ун-та, 2009. – 84 с.
ISBN 978-5-7103-20

В пособии рассматриваются возможности сервера сценариев Windows Script Host, позволяющего непосредственно из операционной системы выполнять сценарии на языках VBScript и JScript, а также использовать внешние серверы автоматизации.

Предназначено для студентов специальностей "Математика" и "Прикладная математика и информатика".

УДК 004.384 (075.8)
ББК 397

Учебное издание

ПОПОВ Андрей Владимирович

СЦЕНАРИИ WINDOWS SCRIPT HOST

Учебное пособие

Печатается в авторской редакции в соответствии с представленным оригинал-макетом

Дизайн обложки

Подписано в печать 00.06.09. Формат 60×84 $\frac{1}{16}$. Усл. печ. л. 4,88.
Тираж 200 экз. Заказ №

Издательство Мордовского университета
Типография Мордовского университета
430005, г. Саранск, ул. Советская, 24

ISBN 978-5-7103-20

Лекция 1. Эволюция инструментов для автоматизации работы в Microsoft Windows

В настоящее время графический интерфейс Windows стал настолько привычным, что многие пользователи и начинающие администраторы даже не задумываются (а часто и просто не знают) об альтернативных способах управления данной операционной системой, связанных с *командной строкой* (command line) и различными *сценариями* (scripts), о тех преимуществах, которые дают эти инструменты с точки зрения *автоматизации работы*, то есть решения различных задач в автоматическом режиме, без участия человека. Подобная ситуация обусловлена тем, что исторически командная строка всегда была слабым местом операционной системы Windows (по сравнению с Unix-системами). Причиной этого в основном является то, что изначально компания Microsoft ориентировалась на широкую аудиторию неискушенных пользователей, не желающих особо вникать в технические детали выполнения тех или иных действий в системе. Поэтому основные усилия разработчиков операционной системы направлялись на улучшение графической оболочки для более комфортной работы непрофессионалов, а не на создание рабочей среды для специалистов или опытных пользователей.

Как показало время, с коммерческой точки зрения на рынке персональных (домашних или офисных) компьютеров эта стратегия оказалась более чем успешной – миллионы людей используют графический интерфейс Windows для запуска нужных им программ, работы в офисных пакетах, просмотра фильмов и т.п. Да и управлять одним Windows-сервером сегодня несложно – операционная система предлагает удобные графические средства для настройки различных параметров и выполнения ежедневных администраторских задач, а с помощью службы терминалов легко можно работать на удаленном сервере, физически расположенном на другом континенте.

Однако подобная модель управления не является масштабируемой: если администрировать не один, а десять серверов, используя стандартные графические инструменты, то одну и ту же последовательность изменения элементов управления в диалоговых окнах придется повторить десять раз, следовательно, в этом случае остро встает вопрос об автоматизации выполнения рутинных операций (например, проведение инвентаризации оборудования и программного обеспечения, мониторинг работы служб, анализ журналов событий и т.д.) на множестве компьютеров. Помочь в этом могут либо специальные (как правило, тяжеловесные и недешевые) приложения типа Microsoft Systems Management Server (SMS), либо сценарии, которые пишутся администраторами самостоятельно (на языке оболочки командной строки или на специальных языках сценариев) и поддерживаются непосредственно операционной системой, без установки сторонних программных продуктов.

Поэтому для профессионала, занимающегося администрированием информационных систем на базе Windows, знание возможностей командной строки, сценариев и технологий автоматизации, поддерживаемых данной операционной системой, просто необходимо.

При этом, однако, неправильно было бы думать, что командная строка или сценарии нужны только администраторам. Ведь ежедневные рутинные задачи пользователей (связанные, например, с копированием или архивированием файлов, подключением или отключением сетевых ресурсов и т.п.), которые обычно выполняются с помощью графического интерфейса проводника Windows, можно полностью самостоятельно автоматизировать, написав нехитрый командный файл, состоящий всего из нескольких строчек! Более того, для человека, не знающего основные команды Windows и такие базовые возможности операционной системы, как перенаправление ввода/вывода и конвейеризация команд, некоторые простейшие задачи могут показаться нетривиальными. Попробуйте, например, пользуясь только графическими средствами, сформировать файл, содержащий имена файлов из всех подкаталогов какого-либо каталога! А ведь для этого достаточно выполнить единственную команду `DIR` (с определенными ключами) и перенаправить вывод этой команды в нужный текстовый файл.

Каким же нам хотелось бы видеть инструмент для автоматизации работы в операционной системе, какими возможностями он должен обладать? Желательно, чтобы в нем было реализовано следующее:

- работа в разных версиях операционной системы (желательно во всех) без установки какого-либо дополнительного программного обеспечения;
- интеграция с командной строкой (непосредственное выполнение вводимых с клавиатуры команд);
- согласованный и непротиворечивый синтаксис команд и утилит;
- наличие подробной встроенной справки по командам с примерами использования;
- возможность выполнения сценариев, составленных на простом для изучения языке;
- возможность использования всех технологий, поддерживаемых операционной системой.

В Unix-системах в качестве инструмента автоматизации выступает стандартная оболочка (shell) или ее модификации (bashell, kshell, cshell и т.д.), причем этот аспект операционной системы стандартизирован в рамках POSIX (стандарт мобильных систем).

В операционной системе Windows дело обстоит сложнее. На сегодняшний день одного "идеального" средства автоматизации, удовлетворяющего сразу всем перечисленным выше требованиям, в Windows нет; в последних версиях операционной системы поддерживаются несколько стандартных инструментов автоматизации, сильно отличающихся друг от друга: оболочка командной строки cmd.exe, среда выполнения сценариев Windows Script Host и оболочка Microsoft PowerShell. Поэтому администратору или пользователю Windows приходится выбирать, каким

именно подходом воспользоваться для решения определенной задачи, а для этого желательно иметь четкое представление о сильных и слабых сторонах данных средств автоматизации. Здесь мы кратко обсудим достоинства и недостатки каждого из них (теоретические и практические подробности будут рассмотрены в последующих лекциях).

Оболочка командной строки command.com/cmd.exe

Во всех версиях операционной системы Windows поддерживается интерактивная *оболочка командной строки* (command shell) и по умолчанию устанавливается определенный набор утилит командной строки (количество и состав этих утилит зависит от версии операционной системы). Вообще, любую операционную систему можно представить в виде совокупности *ядра системы*, которое имеет доступ к аппаратуре и оперирует файлами и процессами, и *оболочки* (*командного интерпретатора*) с утилитами, которые позволяют пользователю получить доступ к функциональности ядра операционной системы. Механизм работы оболочек в разных системах одинаков: в ответ на приглашение ("подсказку", prompt), выдаваемое находящейся в ожидании оболочкой, пользователь вводит некоторую команду (функциональность этой команды может быть реализована либо самой оболочкой, либо определенной внешней утилитой), оболочка выполняет ее, при необходимости выводя на экран какую-либо информацию, после чего снова выводит приглашение и ожидает ввода следующей команды. С технической точки зрения оболочка представляет собой построчный интерпретатор простого языка сентенциального (директивного) программирования, в качестве операторов которого могут использоваться исполняемые программы.

Наряду с интерактивным режимом работы оболочки, как правило, поддерживают и пакетный режим, в котором система последовательно выполняет команды, записанные в текстовом файле-сценарии. Оболочка Windows не является исключением, с точки зрения программирования язык командных файлов Windows может быть охарактеризован следующим образом:

- реализация сентенциальной (директивной) парадигмы программирования;
- выполнение в режиме построчной интерпретации;
- наличие управляющих конструкций;
- поддержка нескольких видов циклов (в том числе специальных циклов для обработки текстовых файлов);
- наличие оператора присваивания (установки значения переменной);
- возможность использования внешних программ (команд) операционной системы в качестве операторов и обработки их кодов возврата;
- наличие нетипизированных переменных, которые декларируются первым упоминанием (значение переменных могут интерпретироваться как числа и использоваться в выражениях целочисленной арифметики).

Начиная с версии Windows NT, оболочка командной строки представляется интерпретатором Cmd.exe, который расширяет возможности

оболочки command.com операционной системы MS-DOS. В свою очередь функциональность командного интерпретатора command.com была позаимствована из операционной системы CP/M, оболочка которой представляла собой значительно упрощенный и урезанный вариант оболочки Unix-систем.

Таким образом, оболочка командной строки MS-DOS изначально уступала Unix-оболочкам по удобству работы и развитости языка сценариев; в командной оболочке Windows (cmd.exe), несмотря на все сделанные улучшения, не удалось преодолеть данное отставание ни в режиме интерактивной работы (например, в cmd.exe отсутствует поддержка псевдонимов для длинных названий команд и не реализовано автоматическое завершение команд при вводе их с клавиатуры), ни в синтаксисе или возможностях языка командных файлов. Ситуация усугублялась тем, что Windows всегда проигрывала Unix-системам в количестве и функциональных возможностях стандартных (не требующих дополнительной установки) утилит командной строки, а также в качестве и полноте встроенной справочной системы по командам оболочки.

На практике проблему отсутствия нужной функциональности у стандартных команд приходится решать либо с помощью утилит пакета Windows Resource Kit для соответствующей версии операционной системы, либо путем поиска подходящей утилиты сторонних производителей. Кроме того, в Windows можно пользоваться POSIX-совместимыми утилитами и оболочками с помощью пакета Microsoft Services For Unix (SFU). Данный продукт разрабатывался еще для Windows NT и первоначально не входил в состав операционной системы, его нужно было приобретать за отдельную плату. В дальнейшем пакет SFU стал бесплатным и даже был включен в состав операционной системы Windows Server 2003 R2.

Итак, учитывая все сказанное выше, мы можем сделать следующий вывод: оболочка командной строки cmd.exe и командные файлы – наиболее универсальные и простые в изучении средства автоматизации работы в Windows, доступные во всех версиях операционной системы, которые, однако, существенно проигрывают аналогичным инструментам в Unix-системах и не обеспечивают доступ к объектным моделям, поддерживаемым операционной системой (COM, WMI, .NET).

Поддержка языков сценариев. Сервер сценариев Windows Script Host

Следующим шагом в развитии средств и технологий автоматизации в операционной системе Windows стало появление сервера сценариев Windows Script Host (WSH) [1]–[4]. Этот инструмент разработан для всех версий Windows и позволяет непосредственно в операционной системе выполнять сценарии на полноценных языках сценариев (по умолчанию, VBScript и JScript), которые до этого были доступны только внутри HTML-страниц и работали в контексте безопасности веб-браузера (в силу этого подобные сценарии, например, могли не иметь доступа к файловой системе локального компьютера).

По сравнению с командными файлами интерпретатора cmd.exe сценарии WSH имеют несколько преимуществ.

Во-первых, VBScript и JScript – это полноценные алгоритмические языки, имеющие встроенные функции и методы для обработки символьных строк, выполнения математических операций, обработки исключительных ситуаций и т.д.; кроме того, для написания сценариев WSH может использоваться любой другой язык сценариев (например, широко распространенный в Unix-системах Perl), для которого установлен соответствующий модуль поддержки.

Во-вторых, WSH поддерживает несколько собственных объектов, свойства и методы которых позволяют решать некоторые часто возникающие повседневные задачи администратора операционной системы: работа с сетевыми ресурсами, переменными среды, системным реестром, ярлыками и специальными папками Windows, запуск и управление работой других приложений.

В-третьих, из сценариев WSH можно обращаться к службам любых приложений-серверов автоматизации (например, программ из пакета Microsoft Office), которые регистрируют в операционной системе свои объекты.

Наконец, сценарии WSH позволяют работать с объектами информационной модели Windows Management Instrumentation (WMI), обеспечивающей программный интерфейс управления всеми компонентами операционной модели, а также с объектами службы каталогов Active Directory Service Interface (ADSI) (объектные модели WMI и ADSI будут обсуждаться подробнее в следующих лекциях).

Следует также отметить, что технология WSH поддерживается в Windows уже довольно давно, в Интернете (в том числе на сайте Microsoft) можно найти множество готовых сценариев, выполняющих ту или иную операцию и при определенных навыке и знаниях быстро "подогнать" эти сценарии под свои конкретные задачи.

Поговорим теперь о слабых местах WSH. Прежде всего, сам по себе WSH – это только среда выполнения сценариев, а не оболочка; WSH не интегрирован с командной строкой, то есть отсутствует режим, в котором можно было вводить команды с клавиатуры и сразу видеть результат их выполнения.

Большим минусом является то, что в операционной системе по умолчанию нет полноценной подробной справочной информации по объектам WSH и языкам VBScript/JScript (документацию приходится искать в Интернете на сайте Microsoft). Другими словами, если вы, например, не помните синтаксис определенной команды VBScript/JScript или точное название свойства объекта WSH, под рукой у вас нет распечатанной документации, а компьютер не имеет выхода в Интернет, то написать корректный сценарий вам просто не удастся (в данном аспекте командные файлы более универсальны, так как практически у всех команд есть по крайней мере встроенное описание используемых ими ключей, а в

операционной системе имеется справочный файл с информацией о всех стандартных командах).

Наконец, сценарии WSH представляют собой довольно серьезную потенциальную угрозу с точки зрения безопасности, известно большое количество вирусов, использующих WSH для выполнения деструктивных действий.

Таким образом, можно дать следующую общую оценку: сценарии WSH – это универсальный инструмент, который в любой версии операционной системы Windows позволяет решать задачи автоматизации практически любой степени сложности, но требует при этом большой работы по изучению самих языков сценариев и ряда смежных технологий управления операционной системой (WMI, ADSI и т.п.).

Командная оболочка Microsoft PowerShell

Итак, к началу XXI века ситуацию со средствами автоматизации работы в Windows нельзя было назвать совсем хорошей. С одной стороны функциональности и гибкости языка оболочки cmd.exe было явно недостаточно, а с другой стороны сценарии WSH, работающие с объектными моделями ADSI и WMI, оказались слишком сложными для пользователей среднего уровня и начинающих администраторов.

В 2000 году была начата разработка новой оболочки для доступа к объектам WMI из командной строки (WMI Command-line, WMIC). Этот продукт оказался не особенно удачным, так как в нем акцент был сделан на функциональные особенности WMI, а не на удобство работы пользователя. Начав дорабатывать WMIC, специалисты Microsoft поняли, что можно реализовать оболочку, которая не ограничивалась бы только работой с объектами WMI, а также предоставляла бы доступ к любым классам платформы .NET Framework, обеспечивая тем самым возможность пользоваться из командной строки всеми мощными функциональными возможностями данной среды.

Перед разработчиками новой оболочки, получившей название Windows PowerShell, стояли следующие основные цели и задачи, которые были успешно решены:

- применение командной строки в качестве основного интерфейса администрирования;
- реализация модели ObjectFlow (элементом обмена информации является объект);
- переработка существующих команд, утилит и оболочки;
- интеграция командной строки, объектов COM, WMI и .NET;
- работа с произвольными источниками данных в командной строке по принципу файловой системы.

Вообще, самая важная идея, заложенная в PowerShell, состоит в том, что в командной строке вывод результатов команды представляет собой не текст (в смысле последовательности байтов), а объект (данные вместе со свойственными им методами). В силу этого работать в PowerShell становится

проще, чем в традиционных оболочках, так как не нужно выполнять никаких манипуляций по выделению нужной информации из символьного потока.

Отметим, что PowerShell одновременно является и оболочкой командной строки (пользователь работает в интерактивном режиме) и средой выполнения сценариев, которые пишутся на специальном языке PowerShell.

Интерактивный сеанс в PowerShell похож на работу в оболочке Unix-систем: все команды в PowerShell имеют подробную встроенную справку (для большинства команд приводятся примеры их использования), поддерживается функция автоматического завершения названий команд и их параметров при вводе с клавиатуры, для многих команд имеются псевдонимы, аналогичные названиям Unix-утилит (`ls`, `pwd`, `tee` и т.д.).

В целом, оболочка PowerShell намного удобнее и мощнее своих предшественников (cmd.exe и WSH), а основным недостатком, сдерживающим распространение нового инструмента, является тот факт, что PowerShell работает не во всех версиях операционной системы Windows. Оболочкой можно пользоваться только на версиях не ниже Windows XP Service Pack 2 с установленным пакетом .NET Framework 2.0.

Контрольные вопросы

Вопрос 1.

Вариант 1. В каких версиях операционной системы Windows можно пользоваться командными файлами?

a. во всех версиях Windows
b. в Windows NT и выше
c. в Windows XP и выше

Вариант 2. Какие версии операционной системы Windows поддерживают сервер сценариев WSH?

a. Windows 2000 и выше
b. все 32-разрядные версии Windows
c. Windows NT и выше

Вопрос 2.

Вариант 1. Какие языки можно использовать для написания сценариев WSH?

a. Microsoft VBScript
b. Microsoft JScript
c. Microsoft C#

Вариант 2. Можно ли написать сценарий WSH на языке Perl?

a. нет
b. да, Perl поддерживается по умолчанию
c. да, но требуется установка специального модуля поддержки Perl

---

Вопрос 3.

Вариант 1. Какое средство автоматизации предлагает собственную объектную модель?

a. Cmd.exe
b. Windows PowerShell
c. WSH

Вариант 2. Можно ли написать сценарий WSH на языке Python?

a. нет
b. да, Python поддерживается по умолчанию
c. да, но только после установки модуля поддержки Python

Лекция 2. Сервер сценариев WSH. Языки сценариев VBScript и JScript

Продолжительное время в качестве стандартного инструмента для автоматизации работы в Windows предлагался только язык командных файлов (язык командной оболочки). Однако с помощью командного интерпретатора cmd.exe трудно написать какую-либо сложную программу-сценарий (script): отсутствует полноценная интерактивность, нельзя напрямую работать с рабочим столом Windows и системным реестром и т. д.

Для исправления этой ситуации компанией Microsoft был разработан сервер сценариев Windows Script Host (WSH), с помощью которого можно выполнять сценарии, написанные, в принципе, на любом языке (при условии, что для этого языка установлен соответствующий модуль (scripting engine), поддерживающий технологию ActiveX Scripting). В качестве стандартных языков поддерживаются Visual Basic Script Edition (VBScript) и JScript.

Вообще говоря, принцип работы сценариев, поддерживаемых WSH, состоит в использовании *объектов ActiveX*, поэтому вначале мы очень кратко опишем возможности самой технологии ActiveX компании Microsoft.

Возможности технологии ActiveX

Напомним, что в Windows с самого начала для обеспечения обмена данными между приложениями была разработана технология *связывания и внедрения объектов* (Object Linking and Embedding, OLE). Вначале технология OLE использовалась для создания составных документов, а затем для решения более общей задачи — предоставления приложениями друг другу собственных функций (служб) и правильного использования этих функций. Технология, позволяющая одному приложению (*клиенту автоматизации*) вызывать функции другого приложения (*сервера автоматизации*) была названа OLE Automation. В основе OLE и OLE Automation лежит разработанная Microsoft базовая "компонентная" технология Component Object Model (COM). В общих словах, компонентное программное обеспечение — это способ разработки программ, при котором используются технологии создания программных модулей, подобные технологиям, применяемым для разработки аппаратных средств. Сложные элементные схемы собираются из стандартизированных микросхем, которые имеют четко определенные документированные функции. Разработчик может эффективно пользоваться такими микросхемами, не задумываясь об их внутренней структуре. В программных компонентах, написанных на каком-либо языке программирования, детали реализации используемых алгоритмов также скрыты внутри компонента (объекта), а на поверхности находятся общедоступные *интерфейсы*, которыми могут пользоваться и другие приложения, написанные на том же или другом языке.

В настоящее время термин OLE используется только по историческим причинам. Вместо него Microsoft с 1996 года использует новый термин — *ActiveX*, первоначально обозначавший WWW (World Wide Web) компоненты (объекты), созданные на базе технологии COM.

Технология ActiveX длительное время являлась ключевой в продуктах Microsoft. Наиболее полное воплощение она нашла в программах Microsoft Office, Internet Explorer, Internet Information Service (IIS). В эти продукты для управления соответствующими объектами автоматизации были встроены интерпретаторы специальных языков сценариев: VBScript (используется в Microsoft Office, Internet Explorer, IIS) и JScript (используется в Internet Explorer, IIS). Однако непосредственно в операционной системе, вне этих продуктов, выполнять сценарии, написанные на VBScript или JScript, было нельзя.

Сервер сценариев WSH является мощным инструментом, предоставляющим единый интерфейс (объектную модель) для специализированных языков (VBScript, JScript, PerlScript, REXX, TCL, Python и т. п.), которые, в свою очередь, позволяют использовать любые внешние объекты ActiveX. С помощью WSH сценарии могут быть выполнены непосредственно в операционной системе Windows, без встраивания в HTML-страницы.

Назначение и основные свойства WSH

WSH предъявляет минимальные требования к объему оперативной памяти, и является очень удобным инструментом для автоматизации повседневных задач пользователей и администраторов операционной системы Windows. Используя сценарии WSH, можно непосредственно работать с файловой системой компьютера, а также управлять работой других приложений (серверов автоматизации). При этом возможности сценариев ограничены только средствами, которые предоставляют доступные серверы автоматизации.

Перечислим только наиболее очевидные задачи, для автоматизации которых прекрасно подходят сценарии WSH.

- Организация резервного копирования на сетевой сервер файлов с локальной машины, которые отбираются по какому-либо критерию.
- Быстрое изменение конфигурации рабочего стола Windows в зависимости от задач, выполняемых пользователем.
- Автоматический запуск программ Microsoft Office, создание там сложных составных документов, распечатка этих документов и закрытие приложений.
- Управление работой приложений, не являющихся серверами автоматизации, с помощью посылки в эти приложения нажатий клавиш.
- Подключение и отключение сетевых ресурсов (дисков и принтеров).
- Создание сложных сценариев регистрации для пользователей.
- Выполнение задач администрирования локальной сети (например, добавление или удаление пользователей).

Создание и запуск простейших сценариев WSH

Простейший WSH-сценарий, написанный на языке JScript или VBScript — это обычный текстовый файл с расширением `js` или `vbs`

соответственно, создать его можно в любом текстовом редакторе, способном сохранять документы в формате "Только текст".

Размер сценария может изменяться от одной до тысяч строк, предельный размер ограничивается лишь максимальным размером файла в соответствующей файловой системе.

В качестве первого примера создадим JScript-сценарий, выводящий на экран диалоговое окно с надписью "Привет!". Для этого достаточно с помощью, например, стандартного Блокнота Windows (notepad.exe) создать файл First.js, содержащий всего одну строку:

```
WScript.Echo("Привет!");
```

Тот же самый сценарий на языке VBScript, естественно, отличается синтаксисом и выглядит следующим образом:

```
WScript.Echo "Привет!"
```

Несмотря на то, что для работы этих двух сценариев достаточно всего одной строки, желательно сразу приучить себя к добавлению в начало файла информации о находящемся в нем сценарии: имя файла, используемый язык, краткое описание выполняемых действий. На языке JScript такая информация, оформленная в виде комментариев, может выглядеть следующим образом:

```
/*****************************************************************/
/* Имя: First.js                                                 */
/* Язык: JScript                                                 */
/* Описание: Вывод на экран приветствия                          */
/*****************************************************************/
```

На языке VBScript то же самое выглядит следующим образом:

```
'****************************************************************
' Имя: First.vbs
' Язык: VBScript
' Описание: Вывод на экран приветствия
'****************************************************************
```

Для запуска сценариев WSH существует несколько способов.

Запуск сценария из командной строки в консольном режиме

Можно выполнить сценарий из командной строки с помощью консольной версии WSH cscript.exe. Например, чтобы запустить сценарий, записанный в файле `C:\Script\First.js`, нужно загрузить командное окно и выполнить в нем команду

```
cscript C:\Script\First.js
```

В результате выполнения этого сценария в командное окно выведется строка "Привет!"

Запуск сценария из командной строки в графическом режиме

Сценарий можно выполнить из командной строки с помощью (оконной) графической версии WSH wscript.exe. Для нашего примера в этом случае нужно выполнить команду

```
wscript C:\Script\First.js
```

Тогда в результате выполнения сценария на экране появится нужное нам диалоговое окно.

Таким образом, мы видим, что при запуске сценария в консольном режиме, вывод текстовой информации происходит в стандартный выходной поток (на экран), при запуске в графическом режиме — в диалоговое окно.

Запуск сценария с помощью меню *Пуск*

Для запуска сценария с помощью пункта **Выполнить** (Run) меню **Пуск** (Start), достаточно написать полное имя этого сценария в поле **Открыть** (Open) и нажать кнопку **Ok**. В этом случае по умолчанию сценарий будет выполнен с помощью wscript.exe, т. е. вывод информации будет вестись в графическое диалоговое окно.

Запуск сценария с помощью Проводника Windows (Windows Explorer)

Самым простым является запуск сценария в окнах Проводника Windows или на рабочем столе — достаточно просто выполнить двойной щелчок мышью на имени файла со сценарием или на его значке (аналогично любому другому исполняемому файлу). При этом, как и в случае запуска с помощью меню **Пуск** (Start), сценарий по умолчанию выполняется с помощью wscript.exe.

Установка и изменение свойств сценариев

В случае необходимости для сценариев можно задавать различные параметры, влияющие на ход их выполнения. Для консольной (cscript.exe) и графической (wscript.exe) версий сервера сценариев эти параметры задаются по-разному.

Если сценарий запускается в консольном режиме, то его исполнение контролируется с помощью параметровкомандной строки для cscript.exe (см. табл. 2.1), которые включают или отключают различные опции WSH (все эти параметры начинаются с символов "`//`").

Таблица 2.1. *Параметры командной строки для cscript.exe*

Параметр	Описание
`//I`	Выключает пакетный режим (по умолчанию). При этом на экран будут выводиться все сообщения об ошибках в сценарии
`//B`	Включает пакетный режим. При этом на экран не будут выводиться никакие сообщения
`//T:nn`	Задает тайм-аут в секундах, т. е. сценарий будет выполняться `nn` секунд, после чего процесс прервется. По умолчанию время выполнения не ограничено
`//Logo`	Выводит (по умолчанию) перед выполнением сценария информацию о версии и разработчике WSH
`//Nologo`	Подавляет вывод информации о версии и разработчике WSH
`//H:CScipt` или `//H:Wscript`	Делает cscript.exe или wscript.exe приложением для запуска сценариев по умолчанию. Если эти параметры не указаны, то по умолчанию подразумевается wscript.exe

//S	Сохраняет установки командной строки для текущего пользователя
//?	Выводит встроенную подсказку для параметров командной строки
//E:engine	Выполняет сценарий с помощью модуля, заданного параметром engine
//D	Включает отладчик
//X	Выполняет программу в отладчике
//Job:<JobID>	Запускает задание с индексом JobID из многозадачного WS-файла (структура WS-файлов будет описана ниже)
//U	Позволяет использовать при перенаправлении ввода-вывода с консоли кодировку Unicode

Например, команда

```
cscript //Nologo C:\Script\First.js
```

запустит сценарий First.js без информации о версии WSH.

Сценарий можно запускать с параметрами командной строки, которые указываются после имени этого сценария (процедура обработки таких параметров будет описана ниже, при рассмотрении объектов `WshArguments`, `WshNamed` и `WshUnnamed`). Например, команда

```
cscript //B C:\Script\First.js /a /b
```

запустит сценарий First.js в пакетном режиме, при этом `/a` и `/b` будут являться параметрами этого сценария, а `//B` — параметром приложения cscript.exe.

Языки VBScript и JScript для сценариев WSH

Как уже отмечалось выше, в принципе сценарии WSH можно писать на любом языке, поддерживающем технологию ActiveX Scripting. Однако для всех таких языков, кроме VBScript и JScript, в системе придется дополнительно устанавливать те или иные модули (библиотеки) поддержки. Поэтому чаще всего при работе с WSH пользуются именно языками VBScript или JScript – сценарии на этих языках гарантированно работают на любой машине с операционной системой Windows.

По синтаксису и стилю программирования эти два языка сильно отличаются друг от друга. Язык JScript — это разработанный Microsoft интерпретируемый объектно-ориентированный язык сценариев, который первоначально предназначался для создания динамических HTML-страниц. Отметим, что JScript не является урезанной версией какого-либо другого языка программирования, хотя по синтаксису он похож на языки Java и C. Язык VBScript (Visual Basic Script Edition) – это облегченная версия языка Microsoft Visual Basic, поэтому для тех, кто программировал на Visual Basic или VBA (Visual Basic for Application) язык VBScript окажется очень знакомым.

При выборе языка для написания сценария, обращающегося к внешним объектам (а это делается практически в каждом сценарии Windows), в пользу VBScript можно привести несколько дополнительных аргументов.

Во-первых, VBScript позволяет напрямую в цикле `For Each … Next` перебирать элементы коллекций (такие коллекции часто являются

свойствами внешних объектов), а в JScript для этого приходится использовать вспомогательный объект `Enumerator` и цикл `for` с условием завершения и оператором итерации. Рассмотрим следующий пример. Предположим, что переменная `Folder` является объектом, соответствующим корневому каталогу диска C:, т.е. Folder представляет собой коллекцию, содержащую объекты-файлы, находящиеся в корневом каталоге. Тогда перебор всех этих файлов в VBScript организуется следующим образом:

```
'Создаем коллекцию Files всех файлов в корневом каталоге диска C:
Set Files = Folder.Files
'Перебираем все элементы коллекции Files
For Each File In Files
  'Выделяем имя файла для текущего элемента File коллекции
  s = s & File.Name & vbCrLf
Next
```

Аналогичный код на JScript выглядит так:

```
//Создаем коллекцию файлов
Files=new Enumerator(Folder.Files);
//Цикл по всем файлам
for (; !Files.atEnd(); Files.moveNext())
  //Добавляем строку с именем файла
  s+=Files.item().Name+"\n";
```

Во-вторых, VBScript позволяет напрямую вызывать методы объектов WMI, в то время как в JScript эти методы приходится вызывать с помощью специального объекта `SWbemObject`, а также объектов `InParam` (входящие параметры для вызова метода) и `OutParam` (параметры, формируемые после выполнения метода). Наконец, подавляющее большинство примеров сценариев, которые можно найти в документации Microsoft или сети Интернет, написаны на VBScript, их можно использовать в своих разработках без изменений, не тратя время на перевод кода на другой язык.

Ниже в примерах сценариев мы будем преимущественно использовать VBScript (подробное рассмотрение синтаксиса этого языка выходит за рамки настоящего пособия).

Контрольные вопросы

Вопрос 1.

Вариант 1. Какое имя имеет сервер сценариев WSH для консольного режима?

a. cscript.exe
b. wscript.exe
c. cmd.exe

Вариант 2. Какое имя имеет сервер сценариев WSH для графического режима?

a. cscript.exe
b. wscript.exe
c. cmd.exe

Вопрос 2.

Вариант 1. Какой режим выполнения будет установлен по умолчанию для сценариев WSH после выполнения команды cscript.exe //H:CScript?

a. консольный
b. графический
c. тот режим, который был до выполнения команды

Вариант 2. С помощью какой команды можно в консольном режиме подавить вывод на экран из сценариев сообщений об ошибках?

a. cscript.exe //I
b. cscript.exe //B
c. cscript.exe //S

Вопрос 3.

Вариант 1. Какие языки поддерживает WSH по умолчанию (без установки дополнительных модулей)?

a. VBScript
b. Visual Basic
c. Python

Вариант 2. Какие языки поддерживает WSH по умолчанию (без установки дополнительных модулей)?

a. C#
b. JScript
c. Java

Вопрос 4.

Вариант 1. Имеется сценарий script.vbs. Какое имя будет иметь файл с параметрами WSH для этого сценария?

a. script.ws
b. script.wsh
c. script.ini

Вариант 2. Имеется сценарий script.vbs. Какое имя будет иметь файл с параметрами WSH для этого сценария?

a. myscript.wsh
b. script.ws
c. script.wsh

Лекция 3. Собственная объектная модель WSH

Перейдем к описанию собственной объектной модели Windows Script Host. С помощью внутренних объектов WSH из сценариев можно выполнять следующие основные задачи:

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

В WSH версии 5.6 (стандартная версия в Windows XP) входят перечисленные ниже объекты:

- `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.
- `WshArguments`. Обеспечивает доступ ко всем параметрам командной строки запущенного сценария или ярлыка Windows.
- `WshNamed`. Обеспечивает доступ к именным параметрам командной строки запущенного сценария.
- `WshUnnamed`. Обеспечивает доступ к безымянным параметрам командной строки запущенного сценария.
- `WshShell`. Позволяет запускать независимые процессы, создавать ярлыки, работать с переменными среды, системным реестром и специальными папками Windows.
- `WshSpecialFolders`. Обеспечивает доступ к специальным папкам Windows.
- `WshShortcut`. Позволяет работать с ярлыками Windows.
- `WshUrlShortcut`. Предназначен для работы с ярлыками сетевых ресурсов.

- `WshEnvironment`. Предназначен для просмотра, изменения и удаления переменных среды.
- `WshNetwork`. Используется при работе с локальной сетью: содержит сетевую информацию для локального компьютера, позволяет подключать сетевые диски и принтеры.
- `WshScriptExec`. Позволяет запускать консольные приложения в качестве дочерних процессов, обеспечивает контроль состояния этих приложений и доступ к их стандартным входным и выходным потокам.
- `WshController`. Позволяет запускать сценарии на удаленных машинах.
- `WshRemote`. Позволяет управлять сценарием, запущенным на удаленной машине.
- `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 (этот объект будет подробно рассмотрен в следующей лекции).

Рассмотрим более подробно несколько объектов WSH, которые часто используются в сценариях.

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.

Отметим, что в сценарии WSH объект `WScript` можно использовать сразу, без какого-либо предварительного описания или создания, так как его экземпляр создается сервером сценариев автоматически. Для использования же всех остальных объектов нужно использовать либо метод `CreateObject`, либо определенное свойство другого объекта.

Свойства объекта `WScript` представлены в табл. 3.1.

Таблица 3.1. *Свойства объекта* `WScript`

Свойство	Описание
`Application`	Предоставляет интерфейс `Idispatch` для объекта `WScript`
`Arguments`	Содержит указатель на коллекцию `WshArguments`, содержащую параметры командной строки для исполняемого сценария
`FullName`	Содержит полный путь к исполняемому файлу сервера сценариев (в Windows XP обычно это C:\WINDOWS\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 (в Windows XP 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

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 табл. 3.2.

Таблица 3.2. *Методы для работы с потоками*

Метод	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)

Напомним, что 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.

Методы объекта WScript

Объект `WScript` имеет несколько методов, которые описаны в табл. 3.3.

Таблица 3.3. *Методы объекта* `WScript`

Метод	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

Приведем дополнительные пояснения и примеры использования для методов, приведенных в табл. 3.3.

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 Echo

Параметры Arg1, Arg2, … метода Echo 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.

Метод Sleep

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

Листинг 3.1. Использование метода WScript.Sleep

```
'*********************************************************************
'* Имя: SleepExample.vbs
'* Язык: VBScript
'* Описание: Использование метода WScript.Sleep
'*********************************************************************
WScript.Echo "Сценарий запущен, отдыхаем…"
WScript.Sleep 5000
WScript.Echo "Выполнение сценария завершено"
'*************  Конец ******************************************
```

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`.

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 табл. 3.4.

Таблица 3.4. *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 для Windows NT/2000/XP или к переменным среды текущего командного окна для Windows 9*x*
`SpecialFolders`	Содержит объект `WshSpecialFolders` для доступа к специальным папкам Windows (рабочий стол, меню **Пуск** (Start) и т. д.)

Опишем теперь методы, имеющиеся у объекта `WshShell` (табл. 3.5).

Таблица 3.5. *Методы объекта* `WshShell`

Метод	Описание
`AppActivate(title)`	Активизирует заданное параметром `title` окно приложения. Строка `title` задает название окна (например, "calc" или "notepad") или идентификатор процесса (Process ID, PID)
`CreateShortcut(strPat`	Создает объект `WshShortcut` для связи с ярлыком Windows (расширение `lnk`) или объект `WshUrlShortcut`

hname)	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(intType, strMessage [,strTarget])	Протоколирует события в журнале Windows NT/2000/XP или в файле WSH.log. Целочисленный параметр intType определяет тип сообщения, строка strMessage — текст сообщения. Параметр strTarget может задаваться только в Windows NT/2000/XP, он определяет название системы, в которой протоколируются события (по умолчанию это локальная система). 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, anyValue [,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

Объекты-коллекции

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

- параметры командной строки запущенного сценария или ярлыка Windows (объекты `WshArguments, WshNamed` и `WshUnnamed`);
- значения переменных среды (объект `WshEnvironment`);
- пути к специальным папкам Windows (объект `WshSpecialFolders`).

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`.

С помощью объекта `WshArguments` можно также выделять и отдельно обрабатывать аргументы сценария, у которых имеются имена (например, `/Name:Andrey`) и безымянные аргументы. Ясно, что использование именных параметров более удобно, так как 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:

- просматривать полный набор всех параметров (как именных, так и безымянных) с помощью коллекции `WshArguments`;
- выделить только те параметры, у которых есть имена (именные параметры) с помощью коллекции `WshNamed`;
- выделить только те параметры, у которых нет имен (безымянные параметры) с помощью коллекции `WshUnnamed`.

В листинге 3.2 приведен пример сценария, в котором на экран выводятся общее количество параметров командной строки, количество именных и безымянных аргументов, а также значения каждой из этих групп параметров.

Листинг 3.2. Доступ к параметрам командной строки запущенного сценария (VBScript)

```
'*****************************************************************
*
' Имя: Args.vbs
' Язык: VBScript
' Описание: Работа с аргументами запущенного сценария
'*****************************************************************
Option Explicit

Dim i,Arg,objArgs,s,objNamedArgs,objUnnamedArgs

Set objArgs = WScript.Arguments  ' Создаем объект WshArguments
```

```
' Определяем общее количество аргументов
s="Всего аргументов: " & objArgs.Count() & vbCrLf
For Each Arg In objArgs
  s=s & Arg & vbCrLf  ' Формируем строки со значениями аргументов
Next

Set objUnnamedArgs=objArgs.Unnamed  ' Создаем объект WshUnnamed
' Определяем количество безымянных аргументов
s=s & vbCrLf & "Безымянных аргументов: " & objUnnamedArgs.length &
vbCrLf
For Each Arg In objUnnamedArgs
  ' Формируем строки со значениями безымянных аргументов
  s=s & Arg & vbCrLf
Next

Set objNamedArgs=objArgs.Named  ' Создаем объект WshNamed
' Определяем количество именных аргументов
s=s & vbCrLf & "Именных аргументов: " & objNamedArgs.Length & vbCrLf
' Проверяем, существует ли аргумент /Имя:
If objNamedArgs.Exists("Имя") Then
  s=s & objNamedArgs("Имя") & vbCrLf
End If
' Проверяем, существует ли аргумент /Comp:
If objNamedArgs.Exists("Comp") Then
  s=s & objNamedArgs("Comp") & vbCrLf
End If

WScript.Echo s   ' Выводим сформированные строки
'*************  Конец *******************************************
```

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:

```
Set WshShell=WScript.CreateObject("WScript.Shell")
Set WshSysEnv=WshShell.Environment
Set 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` (листинг 3.3):

Листинг 3.3. Работа с переменными среды (VBScript)

```
'*****************************************************************
*
' Имя: Environment.vbs
' Язык: VBScript
' Описание: Работа с переменными среды
'*****************************************************************
*
Dim WshShell, WshSysEnv
Set WshShell=WScript.CreateObject("WScript.Shell")
Set 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.

Объект WshSpecialFolders

При установке Windows всегда автоматически создаются несколько специальных папок (например, папка для рабочего стола (Desktop) или папка для меню **Пуск** (Start)), путь к которым впоследствии может быть тем или иным способом изменен. Объект `WshSpecialFolders` обеспечивает доступ к коллекции, содержащей пути к специальным папкам Windows; задание путей к таким папкам может понадобиться, например, для создания непосредственно из сценария ярлыков на рабочем столе.

В Windows XP поддерживаются следующие имена специальных папок:

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

- AllUsersStartup.

Объект `WshSpecialFolders` создается с помощью свойства `SpecialFolders` объекта `WshShell`:

```
var WshShell=WScript.CreateObject("WScript.Shell"),
    WshSpecFold=WshShell.SpecialFolders;
```

В листинге 3.4 приведен сценарий, формирующий список всех имеющихся в системе специальных папок.

Листинг 3.4. Формирование списка всех специальных папок (VBScript)

```
'*****************************************************************
' Имя: SpecFold1.vbs
' Язык: VBScript
' Описание: Вывод названий всех специальных папок Windows
'*****************************************************************
Option Explicit
Dim WshShell, WshFldrs, SpecFldr, s  ' Объявляем переменные
' Создаем объект WshShell
Set WshShell = WScript.CreateObject("Wscript.Shell")
' Создаем объект WshSpecialFolders
Set WshFldrs = WshShell.SpecialFolders
s="Список всех специальных папок:" & vbCrLf & vbCrLf
' Перебираем все элементы коллекции WshFldrs
For Each SpecFldr In WshFldrs
  ' Формируем строки с путями к специальным папкам
  s=s & SpecFldr & vbCrLf
Next
WScript.Echo s
'*************  Конец ********************************************/
```

Объект `WshSpecialFolders` также позволяет получить путь к конкретно заданной специальной папке. Например, в сценарии SpecFold2.vbs (листинг 3.5) на экран выводятся пути к папкам рабочего стола (Desktop), избранных ссылок (Favorites) и раздела **Программы** (Programs) меню **Пуск** (Run).

Листинг 3.5. Доступ к определенным специальным папкам (VBScript)

```
'*****************************************************************
' Имя: SpecFold2.vbs
' Язык: VBScript
' Описание: Вывод названий заданных специальных папок Windows
'*****************************************************************
Option Explicit
Dim WshShell, WshFldrs, s   ' Объявляем переменные
' Создаем объект WshShell
Set WshShell = WScript.CreateObject("Wscript.Shell")
' Создаем объект WshSpecialFolders
Set WshFldrs = WshShell.SpecialFolders
```

```
' Формируем строки с путями к конкретным специальным папкам
s="Некоторые специальные папки:" & vbCrLf & vbCrLf
s=s+"Desktop:"+WshFldrs("Desktop") & vbCrLf
s=s+"Favorites:"+WshFldrs("Favorites") & vbCrLf
s=s+"Programs:"+WshFldrs("Programs")
WScript.Echo s   ' Выводим сформированные строки на экран
'*************  Конец *****************************************/
```

Контрольные вопросы

Вопрос 1.

Вариант 1. Какой из объектов WSH позволяет получить доступ к стандартным потокам ввода/вывода (StdIn/StdOut)?

a. WshEnvironment
b. WshShell
c. WScript

Вариант 2. Каким образом в сценарии создается экземпляр объекта WScript?

a. создается автоматически
b. с помощью функции CreateObject языка VBScript
c. с помощью оператора new языка JScript

Вопрос 2.

Вариант 1. Какой из объектов WSH позволяет получить доступ к переменным среды?

a. WshEnvironment
b. WshNetwork
c. WScript

Вариант 2. Какой из объектов WSH позволяет получить доступ к специальным папкам Windows?

a. WshEnvironment
b. WshSpecialFolders
c. WScript

Вопрос 3.

Вариант 1. Какой из объектов WSH позволяет создавать ярлыки для программ и документов?

a. WshEnvironment
b. WshShell
c. WScript

Вариант 2. Какой из объектов WSH позволяет протоколировать действия в журнале событий Windows?

a. WshShell
b. WshEnvironment
c. WScript

Вопрос 4.

Вариант 1. Каким образом можно из сценария создать экземпляр внешнего объекта-сервера автоматизации?

a. с помощью метода WScript.CreateObject
b. с помощью функции CreateObject языкаVBScript
c. с помощью оператора new языка JScript

Вариант 2. Какой из методов объекта WScript позволяет принудительно завершить работу сценариия?

a. Quit
b. Exit
c. End

Лекция 4. Сценарии WSH для доступа к файловой системе. Объектная модель FileSystemObject

Сценарии WSH позволяют получить полный доступ к файловой системе компьютера, в отличие от JScript- или VBScript-сценариев, внедренных в HTML-страницы, где в зависимости от уровня безопасности, который устанавливается в настройках браузера, те или иные операции могут быть запрещены.

Объекты для основных операций с файловой системой

Для работы с файловой системой из сценариев WSH предназначены восемь объектов, главным из которых является `FileSystemObject`. С помощью методов объекта `FileSystemObject` можно выполнять следующие основные действия:

- копировать или перемещать файлы и каталоги;
- удалять файлы и каталоги;
- создавать каталоги;
- создавать или открывать текстовые файлы;
- создавать объекты `Drive`, `Folder` и `File` для доступа к конкретному диску, каталогу или файлу соответственно.

С помощью свойств объектов `Drive`, `Folder` и `File` можно получить детальную информацию о тех элементах файловой системы, с которыми они ассоциированы. Объекты `Folder` и `File` также предоставляют методы для манипулирования файлами и каталогами (создание, удаление, копирование, перемещение); эти методы в основном копируют соответствующие методы объекта `FileSystemObject`.

Кроме этого имеются три объекта-коллекции: `Drives`, `Folders` и `Files`. Коллекция `Drives` содержит объекты `Drive` для всех имеющихся в системе дисков, `Folders` — объекты `Folder` для всех подкаталогов заданного каталога, `Files` — объекты `File` для всех файлов, находящихся внутри определенного каталога.

Наконец, из сценария можно читать информацию из текстовых файлов и записывать в них данные. Методы для этого предоставляет объект `TextStream`.

В табл. 4.1 кратко описано, какие именно объекты, свойства и методы могут понадобиться для выполнения наиболее часто используемых файловых операций.

Таблица 4.1. *Выполнение основных файловых операций*

Операция	Используемые объекты, свойства и методы
Получение сведений об определенном диске (тип файловой системы, метка тома, общий объем и количество свободного места	Свойства объекта `Drive`. Сам объект `Drive` создается с помощью метода `GetDrive` объекта `FileSystemObject`

и т.д.)

Получение сведений о заданном каталоге или файле (дата создания или последнего доступа, размер, атрибуты и т.д.)	Свойства объектов `Folder` и `File`. Сами эти объекты создаются с помощью методов `GetFolder` и `GetFile` объекта `FileSystemObject`
Проверка существования определенного диска, каталога или файла	Методы `DriveExists`, `FolderExists` и `FileExists` объекта `FileSystemObject`
Копирование файлов и каталогов	Методы `CopyFile` и `CopyFolder` объекта `FileSystemObject`, а также методы `File.Copy` и `Folder.Copy`
Перемещение файлов и каталогов	Методы `MoveFile` и `MoveFolder` объекта `FileSystemObject`, или методы `File.Move` и `Folder.Move`
Удаление файлов и каталогов	Методы `DeleteFile` и `DeleteFolder` объекта `FileSystemObject`, или методы `File.Delete` и `Folder.Delete`
Создание каталога	Методы `FileSystemObject.CreateFolder` или `Folders.Add`
Создание текстового файла	Методы `FileSystemObject.CreateTextFile` или `Folder.CreateTextFile`
Получение списка всех доступных дисков	Коллекция `Drives`, содержащаяся в свойстве `FileSystemObject.Drives`
Получение списка всех подкаталогов заданного каталога	Коллекция `Folders`, содержащаяся в свойстве `Folder.SubFolders`
Получение списка всех файлов заданного каталога	Коллекция `Files`, содержащаяся в свойстве `Folder.Files`
Открытие текстового файла для чтения, записи или добавления	Методы `FileSystemObject.CreateTextFile` или `File.OpenAsTextStream`
Чтение информации из заданного текстового файла или запись ее в него	Методы объекта `TextStream`

Примеры сценариев

Далее приведены простые примеры сценариев, работающих с файловой системой (создание, копирование, удаление файлов и каталогов, чтение и запись строк в текстовом файле и т. д.).

Получение сведений о диске

Доступ к свойствам заданного локального или сетевого диска можно получить с помощью объекта `Drive`, который возвращается методом `GetDrive` объекта `FileSystemObject`, а также может быть получен как элемент коллекции `Drives`.

В листинге 4.1 приведен сценарий DriveInfo.vbs, который выводит на экран некоторые свойства диска C.

Листинг 4.1. Вывод информации о диске (VBScript)

```
'*****************************************************************
' Имя: DriveInfo.vbs
' Язык: VBScript
' Описание: Вывод на экран свойств диска C
'*****************************************************************
'Объявляем переменные
Dim FSO,D,TotalSize,FreeSpace,s
'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
'Создаем объект Drive для диска C
Set D = FSO.GetDrive("C:")
s = "Информация о диске C:" & VbCrLf
'Получаем серийный номер диска
s = s &"Серийный номер: " & D.SerialNumber & VbCrLf
'Получаем метку тома диска
s = s & "Метка тома: " & D.VolumeName & VbCrLf
'Вычисляем общий объем диска в килобайтах
TotalSize = D.TotalSize/1024
s = s & "Объем: " & TotalSize & " Kb" & VbCrLf
'Вычисляем объем свободного пространства диска в килобайтах
FreeSpace = D.FreeSpace/1024
s = s & "Свободно: " & FreeSpace & " Kb" & VbCrLf
'Выводим свойства диска на экран
WScript.Echo s
'*************  Конец ***********************************************
```

Получение сведений о каталоге

Доступ к свойствам каталога обеспечивает объект `Folder`. Создать этот объект можно с помощью свойства `RootFolder` объекта `Drive` или методов `GetFolder`, `GetParentFolder` и `GetSpecialFolder` объекта `FileSystemObject`. Также объекты `Folder` могут быть получены как элементы коллекции `Folders`.

В сценарии FolderInfo.vbs на экран выводятся свойства каталога, из которого был запущен сценарий (листинги 4.2).

Листинг 4.2. Вывод информации о каталоге (VBScript)

```
'*****************************************************************
' Имя: FolderInfo.vbs
' Язык: VBScript
' Описание: Вывод на экран даты создания текущего каталога
'*****************************************************************
Dim FSO,WshShell,FoldSize,s  'Объявляем переменные

'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
'Создаем объект WshShell
```

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

'Определяем каталог, из которого был запущен сценарий
'(текущий каталог)
Set Folder = FSO.GetFolder(WshShell.CurrentDirectory)
'Получаем имя текущего каталога
s = "Текущий каталог: " & Folder.Name & VbCrLf
'Получаем дату создания текущего каталога
s = s & "Дата создания: " & Folder.DateCreated & VbCrLf
'Вычисляем размер текущего каталога в килобайтах
FoldSize=Folder.Size/1024
s = s & "Объем: " & FoldSize & " Kb" & VbCrLf
'Выводим информацию на экран
WScript.Echo s
'*************  Конец ********************************************
```

Получение сведений о файле

Доступ ко всем свойствам файла обеспечивает объект `File`, создать который можно с помощью коллекции `Files` или метода `GetFile` объекта `FileSystemObject`.

В листинге 4.3 приведен сценарий FileInfo.vbs, в котором на экран выводятся некоторые свойства файла C:\boot.ini.

Листинг 4.3. Вывод информации о файле (VBScript)

```
'*****************************************************************
' Имя: FileInfo.vbs
' Язык: VBScript
' Описание: Вывод на экран некоторых свойств файла
'*****************************************************************
Dim FSO,F,s   'Объявляем переменные

'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
'Создаем объект File
Set F = FSO.GetFile("C:\boot.ini")

'Получаем имя файла
s = "Файл: " & F.Name & VbCrLf
'Получаем дату создания файла
s = s & "Дата создания: " & F.DateCreated & VbCrLf
'Получаем тип файла
s = s & "Тип: " & F.Type & VbCrLf
'Выводим информацию на экран
WScript.Echo s
'*************  Конец ********************************************
```

Проверка существования диска, каталога или файла

Для проверки существования диска, каталога и файла используются соответственно методы `DriveExists`, `FolderExists` и `FileExists`. В листинге 4.4 приведен сценарий IsExistsFile.vbs, в котором проверяется наличие на диске C файла boot.ini.

Листинг 4.4. Проверка существования файла (VBScript)

```
'*****************************************************************
' Имя: IsExistsFile.vbs
' Язык: VBScript
' Описание: Проверка существования файла
'*****************************************************************
Dim FSO,FileName 'Объявляем переменные

'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")

FileName = "C:\boot.ini"
if  FSO.FileExists(FileName)  Then
  'Выводим информацию на экран
  WScript.Echo "Файл " & FileName & " существует"
else
  'Выводим информацию на экран
  WScript.Echo "Файл " & FileName & " не существует"
end if
'*************  Конец ***********************************************
```

Получение списка всех имеющихся дисков

Каждому из дисков компьютера (включая подключенные сетевые диски и дисководы со сменными носителями) соответствует элемент коллекции `Drives` (объект `Drive`). Таким образом, для построения списка дисков компьютера нужно в цикле перебрать все элементы коллекции `Drives`.

В листинге 4.5 приведен сценарий ListDrives.vbs, в котором на экран выводятся сведения обо всех доступных дисках.

Листинг 4.5. Построение списка всех имеющихся дисков (VBScript)

```
'*****************************************************************
' Имя: ListDrives.vbs
' Язык: VBScript
' Описание: Получение списка всех имеющихся дисков
'*****************************************************************
'Объявляем переменные
Dim FSO,s,ss,Drives,D

'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
```

```
'Создаем коллекцию дисков, имеющихся в системе
Set Drives = FSO.Drives
s = ""

'Перебираем все диски в коллекции
For Each D In Drives
  'Получаем букву диска
  s = s & D.DriveLetter
  s = s & " - "
  if (D.DriveType = 3) then  'Проверяем, не является ли диск сетевым
    'Получаем имя сетевого ресурса
    ss = D.ShareName
  else
    'Диск является локальным
    if (D.IsReady) then  'Проверяем готовность диска
      'Если диск готов, то получаем метку тома для диска
      ss = D.VolumeName
    else
      ss = "Устройство не готово"
    end if
  s = s & ss & VbCrLf
  end if
Next

'Выводим полученные строки на экран
WScript.Echo s
'*************  Конец *******************************************
```

Получение списка всех подкаталогов заданного каталога

Для построения списка всех подкаталогов определенного каталога можно воспользоваться коллекцией `Folders`, которая хранится в свойстве `SubFolders` соответствующего объекта `Folder` и содержит объекты `Folder` для всех подкаталогов.

В листинге 4.6 приведен сценарий ListSubFold.vbs, в котором на экран выводятся названия всех подкаталогов каталога C:\Program Files.

Листинг 4.6. Построение списка подкаталогов (VBScript)

```
'*********************************************************************
' Имя: ListSubFold.vbs
' Язык: VBScript
' Описание: Получение списка всех подкаталогов заданного каталога
'*********************************************************************
'Объявляем переменные
Dim FSO,F,SFold,SubFolders,Folder,s

'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
```

```
'Путь к каталогу
SFold = "C:\Program Files"
s = "Каталог " & SFold & VbCrLf
s = s & "Подкаталоги:" & VbCrLf
'Создаем объект Folder для каталога C:\Program Files
Set F=FSO.GetFolder(SFold)

'Создаем коллекцию подкаталогов каталога C:\Program Files
Set SubFolders = F.SubFolders

'Цикл по всем подкаталогам
For Each Folder In SubFolders
  'Добавляем строку с именем подкаталога
   s = s & Folder & VbCrLf
Next

'Выводим полученные строки на экран
WScript.Echo s
'*************  Конец *******************************************/
```

Получение списка всех файлов заданного каталога

В свойстве `Files` объекта `Folder`, соответствующего определенному каталогу, хранится коллекция находящихся в этом каталоге файлов (объектов `File`). В листинге 4.7 приведен сценарии ListFiles.vbs, выводящий на экран названия всех файлов, которые содержатся в специальной папке Мои документы.

Листинг 4.7. Построение списка файлов в каталоге (JScript)

```
'*****************************************************************
' Имя: ListFiles.vbs
' Язык: VBScript
' Описание: Получение списка всех файлов заданного каталога
'*****************************************************************
'Объявляем переменные
Dim FSO,F,File,Files,WshShell,PathList,s

'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
'Создаем объект WshShell
Set WshShell = WScript.CreateObject("Wscript.Shell")
'Создаем объект WshSpecialFolders
Set WshFldrs = WshShell.SpecialFolders
'Определяем путь к папке Мои документы
PathList = WshFldrs.item("MyDocuments") & "\"
'Создаем объект Folder для папки Мои документы
Set F = FSO.GetFolder(PathList)
'Создаем коллекцию файлов каталога Мои документы
Set Files = F.Files
```

```
s = "Файлы из каталога " & PathList & VbCrLf
'Цикл по всем файлам
For Each File In Files
  'Добавляем строку с именем файла
  s = s & File.Name & VbCrLf
Next

'Выводим полученные строки на экран
WScript.Echo s
'*************  Конец ********************************************
```

Создание каталога

Создать новый каталог на диске можно либо с помощью метода `CreateFolder` объекта `FileSystemObject`, либо с помощью метода `Add` коллекции `Folders`. Оба эти метода используются в сценарии MakeFolder.vbs для создания в каталоге C:\Мои документы подкаталогов Новая папка и Еще одна новая папка (листинг 4.8).

Листинг 4.8. Создание нового каталога (VBScript)

```
'*****************************************************************
' Имя: MakeFolder.vbs
' Язык: VBScript
' Описание: Создание нового каталога
'*****************************************************************
'Объявляем переменные
Dim FSO, F, SubFolders

'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
'Создаем каталог C:\Program Files\Новая папка
FSO.CreateFolder("C:\Program Files\Новая папка")
'Создаем объект Folder для каталога C:\Program Files
Set F = FSO.GetFolder("C:\Program Files")
'Создаем коллекцию подкаталогов каталога C:\Program Files
Set SubFolders = F.SubFolders
'Создаем каталог C:\Program Files\Еще одна новая папка
SubFolders.Add "Еще одна новая папка"
'*************  Конец ********************************************
```

Создание текстового файла

Для создания текстового файла используется метод `CreateTextFile` объекта `FileSystemObject`, который имеет один обязательный текстовый параметр (путь к создаваемому файлу) и два необязательных логических параметра (`Overwrite` и `Unicode`).

Параметр `Overwrite` имеет значение в том случае, когда создаваемый файл уже существует. Если `Overwrite` равно `True`, то такой файл перепишется (старое содержимое будет утеряно), если же в качестве

Overwrite указано False, то файл переписываться не будет. Если этот параметр вообще не указан, то существующий файл также не будет переписан.

Параметр Unicode указывает, в каком формате (ASCII или Unicode) следует создавать файл. Если Unicode равно True, то файл создается в формате Unicode, если же Unicode равно False или этот параметр вообще не указан, то файл создается в режиме ASCII.

В сценарии CreateTempFile.vbs (листинг 4.9) показано, каким образом можно создать файл со случайно выбранным именем (такие файлы часто используются для записи временных данных).

Листинг 4.9. Создание временного файла со случайным именем (VBScript)

```
'*************************************************************
' Имя: CreateTempFile.vbs
' Язык: VBScript
' Описание: Создание временного файла со случайным именем
'*************************************************************
Dim FSO,FileName,F,s  'Объявляем переменные

'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
'Генерируем случайное имя файла
FileName = FSO.GetTempName
'Создаем файл с именем FileName
Set F = FSO.CreateTextFile(FileName, true)
'Закрываем файл
F.Close
'Сообщаем о создании файла
WScript.Echo "Был создан файл " & FileName
'*************  Конец **********************************************
```

Операции чтения и записи строк для текстового файла

Последовательный (строка за строкой) доступ к текстовому файлу обеспечивает объект TextStream. Методы этого объекта позволяют читать информацию из файла и записывать ее в него. Создается объект TextStream при открытии или создании текстового файла с помощью следующих методов:

- CreateTextFile объектов FileSystemObject и Folder;
- OpenTextFile объекта FileSystemObject;
- OpenAsTextStream объекта File.

Перечисленные ранее методы позволяют открывать файл в разных режимах ввода/вывода (за это отвечает параметр Iomode) с использованием разных форматов (в соответствии со значением параметра Format) (см. табл. 4.2, 4.3)

Таблица 4.2. *Параметр* `Iomode`

Константа	Значение	Описание
`ForReading`	1	Файл открывается только для чтения, записывать информацию в него нельзя
`ForWriting`	2	Файл открывается для записи. Если файл с таким именем уже существовал, то при новой записи его содержимое теряется
`ForAppending`	8	Файл открывается для добавления. Если файл уже существовал, то информация будет дописываться в конец этого файла

Таблица 4.3. *Параметр* `Format`

Константа	Значение	Описание
`TristateUseDefault`	-2	Файл открывается в формате, используемом системой по умолчанию
`TristateTrue`	-1	Файл открывается в формате Unicode
`TristateFalse`	0	Файл открывается в формате ASCII

В листинге 4.10 приведен сценарий TextFile.vbs, в котором создается файл test1.txt и в него записываются строка текста. После этого файл открывается для чтения, строка считывается из него и выводится на экран.

Листинг 4.10. Запись информации в текстовый файл и чтение из него (VBScript)

```
'*****************************************************************
' Имя: TextFile.vbs
' Язык: VBScript
' Описание: Запись строк в текстовый файл и чтение из него
'*****************************************************************
Dim FSO,F,TextStream,s  'Объявляем переменные
' Инициализируем константы
Const ForReading = 1, ForWriting = 2, TristateUseDefault = -2

' Создаем объект FileSystemObject
Set FSO=WScript.CreateObject("Scripting.FileSystemObject")
' Создаем в текущем каталоге файл test1.txt
FSO.CreateTextFile "test1.txt"
' Создаем объект File для файла test1.txt
set F=FSO.GetFile("test1.txt")
' Создаем объект TextStream (файл открывается для записи)
Set TextStream=F.OpenAsTextStream(ForWriting, TristateUseDefault)
' Записываем в файл строку
TextStream.WriteLine "Это первая строка"
' Закрываем файл
TextStream.Close
' Открываем файл для чтения
```

```
Set TextStream=F.OpenAsTextStream(ForReading, TristateUseDefault)
' Считываем строку из файла
s=TextStream.ReadLine
' Закрываем файл
TextStream.Close
' Отображаем строку на экране
WScript.Echo "Первая строка из файла test1.txt:" & vbCrLf & vbCrLf &
s
'*************  Конец ******************************************
```

Копирование и перемещение файлов и каталогов

Для копирования файлов/каталогов можно применять метод `CopyFile/CopyFolder` объекта `FileSystemObject` или метод `Copy` соответствующего этому файлу/каталогу объекта `File/Folder`. Перемещаются файлы/каталоги с помощью методов `MoveFile/MoveFolder` объекта `FileSystemObject` или метода `Move` соответствующего этому файлу/каталогу объекта `File/Folder`.

Отметим, что при использовании всех этих методов процесс копирования или перемещения прерывается после первой возникшей ошибки. Кроме того, нельзя перемещать файлы и каталоги с одного диска на другой.

В листинге 4.11 приведен сценарий CopyFile.vbs, иллюстрирующий использование метода `Copy`. В этом сценарии на диске C создается файл TestFile.txt, который затем копируется на рабочий стол.

Листинг 4.11. Создание текстового файла и копирование его в другой каталог (VBScript)

```
'*********************************************************************
' Имя: CopyFile.vbs
' Язык: VBScript
' Описание: Создание и копирование файла
'*********************************************************************
'Объявляем переменные
Dim FSO,F,WshShell,WshFldrs,PathCopy

'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
'Создаем файл
Set F = FSO.CreateTextFile("C:\TestFile.txt", true)
'Записываем в файл строку
F.WriteLine "Тестовый файл"
'Закрываем файл
F.Close

'Создаем объект WshShell
Set WshShell = WScript.CreateObject("Wscript.Shell")
'Создаем объект WshSpecialFolders
Set WshFldrs = WshShell.SpecialFolders
```

```
'Определяем путь к рабочему столу
PathCopy = WshFldrs.item("Desktop")+"\"
'Создаем объект File для файла C:\TestFile.txt
Set F = FSO.GetFile("C:\TestFile.txt")
'Копируем файл на рабочий стол
F.Copy PathCopy
'*************  Конец *********************************************
```

Удаление файлов и каталогов

Для копирования файлов/каталогов можно применять метод `DeleteFile`/`DeleteFolder` объекта `FileSystemObject` или метод `Delete` соответствующего этому файлу/каталогу объекта `File`/`Folder`. Отметим, что при удалении каталога неважно, является ли он пустым или нет — удаление будет произведено в любом случае. Если же заданный для удаления файл/каталог не будет найден, то возникнет ошибка.

В листинге 4.12 приведен сценарий DeleteFile.vbs, в котором производится удаление предварительно созданного файла C:\TestFile.txt.

Листинг 4.12. Создание и удаление файла (VBScript)

```
'*****************************************************************
' Имя: DeleteFile.vbs
' Язык: VBScript
' Описание: Создание и удаление файла
'*****************************************************************
'Объявляем переменные
Dim FSO,F,FileName

'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
'Задаем имя файла
FileName="C:\TestFile.txt"
'Создаем файл
Set F = FSO.CreateTextFile(FileName, true)
'Записываем в файл строку
F.WriteLine "Тестовый файл"
'Закрываем файл
F.Close
WScript.Echo "Файл создан"
FSO.DeleteFile FileName
WScript.Echo "Файл удален"
'*************  Конец *********************************************
```

Контрольные вопросы

Вопрос 1.

Вариант 1. Какой программный идентификатор (ProgID) имеет объект FileSystemObject?

a. Scripting.FSO
b. WScript.FileSystemObject
c. Scripting.FileSystemObject

Вариант 2. Каким образом в сценарии на языке VBScript можно создать экземпляр объекта FileSystemObject?

a. Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
b. Set FSO = CreateObject("Scripting.FileSystemObject")
c. Set FSO = WScript.CreateObject("WScript.FileSystemObject")

Вопрос 2.

Вариант 1. Пусть D – экземпляр объекта Drive, соответствующий логическому диску D:. В сценарии выполняется следующая строка (язык VBScript): n=D.TotalSize/(1024*1024). Чему после этого равно значение переменной n?

a. общий объем диска D: в килобайтах
b. общий объем диска D: в мегабайтах
c. общий объем диска D: в гигабайтах

Вариант 2. Пусть D – экземпляр объекта Drive, соответствующий логическому диску D:. В сценарии выполняется следующая строка (язык VBScript): n=D.FreeSpace/(1024*1024). Чему после этого равно значение переменной n?

a. объем свободного места на диске D: в килобайтах
b. объем свободного места на диске D: в мегабайтах
c. объем свободного места на диске D: в гигабайтах

Вопрос 3.

Вариант 1. Пусть FSO – экземпляр объекта FileSystemObject. Каким образом можно создать временный текстовый файл со случайным именем (язык VBScript)?

a. F=FSO.CreateTextFile(FSO.GetTempName, true)
b. Set F=FSO.CreateTextFile(FSO.GetTempName, true)
c. Set F=FSO.CreateTextFile(FSO.GetRandomName, true)

Вариант 2. Пусть FSO – экземпляр объекта FileSystemObject. Каким образом можно создать текстовый файл с именем "Мой файл.txt" (язык VBScript)?

a. Set F=FSO.CreateTextFile("Мой файл.txt", true)
b. F=FSO.CreateTextFile("Мой файл.txt", true)
c. Set F=FSO.CreateFile("Мой файл.txt", true)

Вопрос 4.

Вариант 1. Пусть FSO – экземпляр объекта FileSystemObject. Каким образом можно удалить файл C:\1.txt (язык VBScript)?

a. FSO. EraseFile "C:\1.txt"
b. FSO.DeleteFile "C:\1.txt"
c. FSO.Delete "C:\1.txt"

Вариант 2. Какой метод объекта FileSystemObject позволяет удалять файлы?

a. DeleteFile
b. EraseFile
c. Delete

Лекция 5. WSH как средство взаимодействия с внешними программами

Сценарии WSH способны запускать другие графические или консольные приложения, причем в последнем случае запущенный процесс может функционировать как дочерний, что позволяет в сценарии-родителе работать с информацией из его входного или выходного потока.

Запуск приложений Windows

Из сценария WSH запустить другое приложение можно с помощью методов `Run` или `Exec` объекта `WshShell`.

При использовании метода `Run` для запускаемого приложения можно задать тип окна (при условии, что приложение поддерживает этот тип). Например, в результате выполнения следующих двух строк VBScript-кода:

```
Set WshShell = WScript.CreateObject("WScript.Shell")
WshShell.Run "notepad",3
```

программа Блокнот (notepad.exe) будет запущена в максимизированном (развернутом на весь экран) окне.

> ***Замечание***
> Метод `Run` всегда создает новый экземпляр запускаемого процесса, с его помощью нельзя ни повторно активизировать окно запущенного приложения (для этого используется метод `AppActivate`), ни свернуть или развернуть его.

Другим вариантом запуска из сценария приложения Windows, является применение метода `Exec`. Этод метод запускает приложение, путь к которому указан как параметр метода, и возвращает объект `WshScriptExec`. Например:

```
Set WshShell = WScript.CreateObject("WScript.Shell")
Set theNotepad = WshShell.Exec("calc");
```

> ***Замечание***
> При подобном запуске приложения, в отличие от метода `Run`, нельзя задать тип окна.

Объект `WshScriptExec` позволяет контролировать ход выполнения запущенного приложения с помощью свойства `Status` — если `Status` равен `0`, то приложение выполняется, если `Status` равен `1`, то приложение завершено. Кроме этого, используя метод `Terminate`, можно принудительно завершить работу того приложения, которому соответствует объект `WshScriptExec`.

В листинге 5.1 приведен сценарий на языке VBScipt, в котором с помощью метода `Exec` запускается Блокнот (notepad.exe); ссылка на соответствующий объект `WshScriptExec` сохраняется в переменной `theNotepad`:

```
Set theNotepad = WshShell.Exec("notepad")
```

После этого выполнение сценария приостанавливается на 1 секунду (пауза необходима, для того чтобы окно Блокнота успело появиться на экране), после чего выводится диалоговое окно с информацией о статусе

запущенного приложения и вопросом о необходимости закрытия Блокнота. В случае утвердительного ответа происходит закрытие Блокнота с помощью метода `Terminate`.

Листинг 5.1. Запуск и принудительное закрытие приложения (VBScript)

```
'*****************************************************************
' Имя: ExecWinApp.vbs
' Язык: VBScript
' Описание: Запуск и закрытие приложение (объект WshScriptExec)
'*****************************************************************
Option Explicit

Dim WshShell,theNotepad,Res,Text,Title   ' Объявляем переменные
' Создаем объект WshShell
Set WshShell = WScript.CreateObject("WScript.Shell")
WScript.Echo "Запускаем Блокнот"
' Запускаем приложение (создаем объект WshScriptExec)
Set theNotepad = WshShell.Exec("notepad")
WScript.Sleep 500   ' Приостанавливаем выполнение сценария
Text="Блокнот запущен (Status=" & theNotepad.Status & ")" & vbCrLf _
      & "Закрыть Блокнот?"
Title=""
' Выводим диалоговое окно на экран
Res=WshShell.Popup(Text,0,Title,vbQuestion+vbYesNo)
' Определяем, какая кнопка нажата в диалоговом окне
If Res=vbYes Then
  theNotepad.Terminate ' Прерываем работу Блокнота
  ' Приостанавливаем выполнение сценария, для того чтобы Блокнот
  ' успел закрыться
  WScript.Sleep 100
  WScript.Echo "Блокнот закрыт (Status=" & theNotepad.Status & ")"
End If
'*************  Конец ********************************************/
```

Переключение между приложениями, имитация нажатий клавиш

Производить переключение между окнами нескольких запущенных приложений позволяет метод `AppActivate` объекта `WshScript`. В качестве аргумента этого метода нужно указывать либо заголовок активизируемого окна, либо идентификатор процесса (Process ID, PID), который запущен в данном окне. Предпочтительным является использование PID, получаемого с помощью свойства `ProcessID` объекта `WshScriptExec`, соответствующего активизируемому приложению. Недостатки применения заголовка окна в методе `AppActivate`:

- при написании сценария необходимо знать точное название заголовка;
- само приложение может изменить текст в заголовке окна;

- в случае нескольких окон с одинаковыми заголовками `AppActivate` всегда будет активизировать один и тот же экземпляр, доступ к другим окнам получить не удастся.

Активизировав то или иное окно, в котором выполняется приложение Windows, можно из сценария сымитировать нажатия клавиш в этом окне. Для этого используется метод `SendKeys` объекта `WshShell`.

Рассмотрим пример сценария Run&ExecWinApp.vbs (листинг 5.2), в котором запускается Калькулятор (calc.exe) и в его окно с помощью `SendKeys` последовательно посылаются нажатия клавиш <1>, <+>, <2> и <Enter>:

```
Set theCalculator = WshShell.Exec("calc")
WScript.Sleep 500
WshShell.AppActivate theCalculator.ProcessID
WshShell.SendKeys "1{+}"
WshShell.SendKeys "2")
WshShell.SendKeys "~")  ' Клавиша <Enter>
```

Затем выполнение сценария приостанавливается на 1 секунду, чтобы результат вычислений был виден на экране и результат вычислений (символ "3") копируется в буфер Windows с помощью "нажатия" клавиш <Ctrl>+<C>:

```
WshShell.SendKeys "^c"
```

После этого на экран выводится сообщение о том, что Калькулятор будет закрыт, в результате чего окно Калькулятора теряет фокус. Для того чтобы вновь активизировать это окно, используется метод `AppActivate`, параметром которого служит PID Калькулятора:

```
WshShell.AppActivate theCalculator.ProcessID
```

Для того чтобы закрыть окно Калькулятора, в него посылаются нажатия клавиш <Alt>+<F4>:

```
WshShell.SendKeys "%{F4}"
```

После закрытия Калькулятора запускается Блокнот (notepad.exe) и в него записываются результаты работы Калькулятора (вставка из буфера вычисленной суммы производится с помощью "нажатий" <Ctrl>+<V>):

```
WshShell.Run "notepad"
WScript.Sleep 1000
WshShell.AppActivate "notepad"
WshShell.SendKeys "1{+}2="
WshShell.SendKeys "^v"
WshShell.SendKeys " {(}c{)} Calculator"
```

Листинг 5.2. Запуск двух приложений и обмен данными между ними (VBScript)

```
'*****************************************************************
' Имя: Run&ExecWinApp.vbs
' Язык: VBScript
' Описание: Запуск двух приложений и обмен данными между ними
'*****************************************************************
Option Explicit
```

```
Dim WshShell, theCalculator  ' Объявляем переменные
' Создаем объект WshShell
Set WshShell = WScript.CreateObject("WScript.Shell")
WScript.Echo("Запускаем калькулятор и" & vbCrLf & "считаем 1+2")
' Создаем объект WshScript (запускаем Калькулятор)
Set theCalculator = WshShell.Exec("calc")
' Приостанавливаем выполнение сценария, для того чтобы
' окно Калькулятора успело появиться на экране
WScript.Sleep 500
' Активизируем окно Калькулятора
WshShell.AppActivate theCalculator.ProcessID
' Посылаем нажатия клавиш в окно Калькулятора
WshShell.SendKeys "1{+}"
WshShell.SendKeys "2"
WshShell.SendKeys "~"    ' Клавиша <Enter>
WScript.Sleep 500
' Копируем результат вычисления в буфер Windows (<Ctrl>+C)
WshShell.SendKeys "^c"
' Выводим сообщение (активное окно меняется)
WScript.Echo "Закрываем калькулятор"
' Активизируем окно Калькулятора
WshShell.AppActivate theCalculator.ProcessID
' Закрываем окно Калькулятора (<Alt>+<F4>)
WshShell.SendKeys "%{F4}"
WScript.Echo "Запускаем Блокнот и копируем туда результат"
WshShell.Run "notepad"     ' Запускаем Блокнот
' Приостанавливаем выполнение сценария, для того чтобы
' окно Блокнота успело появиться на экране
WScript.Sleep 1000
WshShell.AppActivate "notepad"    ' Активизируем окно Блокнота
' Посылаем нажатия клавиш в окно Блокнота
WshShell.SendKeys "1{+}2="
' Вставляем содержимое буфера Windows (<Ctrl>+V)
WshShell.SendKeys "^v"
' Выводим в окно Блокнота оставшуюся информацию
WshShell.SendKeys " {(}c{)} Calculator"
'*************  Конец ********************************************
```

Запуск независимых консольных приложений и команд Windows

Для запуска независимых, т. е. работающих в отдельном адресном пространстве и использующих свою копию переменных среды, консольных приложений или внешних (представленных исполняемыми файлами на жестком диске) команд Windows используется метод `Run` объекта `WshShell`. При этом выполнение сценария можно приостановить до окончания работы запущенного приложения, а затем проанализировать код выхода этого приложения (для этого третий параметр метода `Run` должен равняться `True`). Соответствующий пример сценария приведен в листинге 5.3.

Листинг 5.3. Запуск независимого консольного приложения (VBScript)

```
'*****************************************************************
' Имя: RunConApp.vbs
' Язык: VBScript
' Описание: Запуск независимого консольного приложения и
'           определение его кода выхода
'*****************************************************************
Option Explicit

Dim WshShell, Code   ' Объявляем переменные
' Создаем объект WshShell
Set WshShell = WScript.CreateObject("WScript.Shell")
' Запускаем утилиту xcopy с ключом "/?" и ожидаем окончания ее
работы
Code=WshShell.Run("xcopy /?",1,true)
' Печатаем полученный код возврата
WScript.Echo "Код возврата: ", Code
'*************  Конец *********************************************/
```

Для выполнения внутренней команды Windows нужно запустить командный интерпретатор cmd.exe и передать ему в качестве параметра нужную команду. Для того чтобы при вызове командного интерпретатора не заботиться о полном пути к cmd.exe, нужно использовать переменную среды `COMSPEC`.

Замечание

Для получения значения переменной среды ее имя нужно окружить знаками `%` (например, `%COMSPEC%`).

В листингах 5.4 приведен сценарий, в котором запускаются внутренние команды `COPY /?` (вызов встроенной справки для `COPY`) и `DIR %WINDIR%` (вывод содержимого системного каталога Windows).

При этом окно, в котором выполняется команда `COPY /?`, не закрывается после завершения этой команды, так как при запуске командного интерпретатора был указан ключ `/k`, а информация, выводимая командой `DIR %WINDIR%`, перенаправляется в файл windir.txt, после чего командное окно закрывается, поскольку для командного интерпретатора в этом случае был указан ключ `/c`.

Листинг 5.4. Запуск внутренней команды Windows (JScript)

```
'*****************************************************************
' Имя: RunDOSCom.vbs
' Язык: VBScript
' Описание: Выполнение внутренних команд Windows
'*****************************************************************
Option Explicit

Dim WshShell, Code   ' Объявляем переменные
```

```
' Создаем объект WshShell
Set WshShell = WScript.CreateObject("WScript.Shell")
' Запускаем внутреннюю команду COPY
WshShell.Run "%COMSPEC% /k copy /?",1
' Запускаем внутреннюю команду DIR
WshShell.Run "%COMSPEC% /c dir %WINDIR% > windir.txt",1
'*************  Конец ******************************************/
```

Запуск дочерних консольных приложений и команд DOS, использование их входных и выходных потоков

Консольное приложение или команду DOS можно запустить из сценария как дочернюю задачу, т. е. с теми же переменными среды, что у процесса-родителя. При этом информация, выводимая дочерним процессом, на экран дублироваться не будет, однако из родительского сценария можно считывать информацию из выходного потока и посылать данные во входной поток дочерней задачи (это напоминает конвейеризацию команд Windows, при которой данные выходного потока одной команды поступают во входной поток другой команды, например `DIR | MORE`). Таким образом, из сценария можно запускать ту или иную утилиту командной строки и обрабатывать выводимые ей данные; иногда таким образом получить нужную информацию бывает проще и быстрее, чем при использовании объектной модели WSH или другого сервера автоматизации.

В качестве примера рассмотрим сценарий ExecConApp.vbs (листинг 5.5), который выводит на экран общее количество файлов в текущем каталоге и их имена. Данную информацию можно получить в сеансе командй строки с помощью команды `DIR /B`. Поэтому в сценарии мы запустим в качестве дочернего процесса внутреннюю команду `DIR` с ключом `/B`:

```
Set ObjExec=WshShell.Exec("%COMSPEC% /c dir /b")
```

и полностью считатем данные, появляющиеся в выходном потоке этого процесса. Для этого в цикле вызывается метод `ReadAll`, считывающий всю информацию, имеющуюся к тому времени в потоке `StdOut` объекта `ObjExec`, в переменную `s`:

```
IsBreak=False
Do While True  ' Бесконечный цикл
  ' Проверяем, достигнут ли конец выходного потока команды DIR
  If (Not ObjExec.StdOut.AtEndOfStream) Then
    ' Считываем полностью выходной поток команды DIR
    s=s+ObjExec.StdOut.ReadAll
  End If
  If IsBreak Then
    Exit Do  ' Выходим из цикла
  End If
  ' Проверяем, не завершилось ли выполнение DIR
  If ObjExec.Status=1 Then
    IsBreak=True
  Else
```

```
    WScript.Sleep 100 ' Приостанавливаем сценарий на 0,1 сек
  End If
Loop
```

Родительский и дочерний процессы работают асинхронно, поэтому пока команда `DIR` не перестанет выдавать данные, т. е. пока свойство `Status` объекта `ObjExec` не станет равным `1`, выполнение сценария с помощью метода `WScript.Sleep` периодически приостанавливается на 0,1 секунды.

После того как считаны все данные из выходного потока команды `DIR` (свойство `ObjExec.StdOut.AtEndOfStream` равно `True`), происходит выход из цикла и формирование из переменной s массива выведенных строк:

```
ArrS=Split(s,vbCrLf)  ' Формируем массив строк
```

Теперь остается подсчитать количество файлов в каталоге, которое на единицу меньше количества строк в массиве `ArrS`:

```
ColFiles=UBound(ArrS)  ' Количество файлов в текущем каталоге
```

и вывести нужные строки на экран:

```
WScript.StdOut.WriteLine "Всего файлов в текущем каталоге: " &
ColFiles
For i=0 To ColFiles-1
  WScript.StdOut.WriteLine ArrS(i)  ' Выводим строки на экран
Next
```

Листинг 5.5. Запуск дочернего консольного приложения (VBScript)

```
'*********************************************************************
' Имя: ExecConApp.vbs
' Язык: VbScript
' Описание: Запуск дочернего консольного приложения
'*********************************************************************
Option Explicit
' Объявляем переменные
Dim ObjExec,WshShell,s,IsBreak,ArrS,ColStr,ColFiles,i
' Создаем объект WshShell
Set WshShell = WScript.CreateObject("WScript.Shell")
' Запускаем команду DIR
Set ObjExec=WshShell.Exec("%COMSPEC% /c dir /b")
s=""
IsBreak=False
Do While True  ' Бесконечный цикл
  ' Проверяем, достигнут ли конец выходного потока команды DIR
  If (Not ObjExec.StdOut.AtEndOfStream) Then
    ' Считываем полностью выходной поток команды DIR
    s=s+ObjExec.StdOut.ReadAll
  End If
  If IsBreak Then
    Exit Do  ' Выходим из цикла
  End If
  ' Проверяем, не завершилось ли выполнение DIR
  If ObjExec.Status=1 Then
```

```
    IsBreak=True
  Else
    WScript.Sleep 100 ' Приостанавливаем сценарий на 0,1 сек
  End If
Loop
ArrS=Split(s,vbCrLf)  ' Формируем массив строк
ColFiles=UBound(ArrS)  ' Количество файлов в текущем каталоге
WScript.StdOut.WriteLine "Всего файлов в текущем каталоге: " &
ColFiles
For i=0 To ColFiles-1
  WScript.StdOut.WriteLine ArrS(i)  ' Выводим строки на экран
Next
'*************  Конец ******************************************
```

Использование внешних серверов автоматизации

Одна из наиболее полезных возможностей сценариев WSH – управление работой внешних серверов автоматизации (программ, предоставляющих свои службы и интерфейсы для использования другими приложениями). Самые распространенные и часто используемые серверы автоматизации в Windows – это приложения пакета Microsoft Office. Мы рассмотрим на примерах, каким образом можно выводить из сценариев WSH информацию в две наиболее распространенные программы этого пакета: Microsoft Word и Microsoft Excel.

Объектные модели Microsoft Word и Excel

Для того, чтобы использовать из сценариев WSH те возможности, которые поддерживают программы Word и Excel, необходимо знать, какие именно объекты предоставляются для внешнего использования этими серверами автоматизации и как объекты соотносятся друг с другом. Хотя объектные модели различных приложений Microsoft Office довольно сложны, они похожи друг на друга, причем для практических целей достаточно понять принцип работы с несколькими ключевыми объектами. Здесь мы не будем останавливаться на подробном рассмотрении свойств и методов объектов Word и Excel, а лишь кратко упомянем, какие именно объекты будут использоваться в рассмотренных ниже примерах сценариев.

На самом верхнем уровне объектной модели Word находится объект `Application`, который представляет непосредственно само приложение `Word` и содержит (в качестве свойств) все остальные объекты. Таким образом, объект `Application` используется для получения доступа к любому другому объекту Word.

Семейство `Documents` является свойством объекта `Application` и содержит набор объектов `Document`, каждый из которых соответствует открытому в Word документу. Класс `Documents` понадобится нам в сценариях для создания новых документов. Объект `Document` содержит в качестве своих свойств семейства различных объектов документа: символов (`Characters`), слов (`Words`), предложений (`Sentences`), параграфов (`Paragraphs`), закладок (`Bookmarks`) и т.д.

Объект `Selection` позволяет работать с выделенным фрагментом текста (этот фрагмент может быть и пустым). Таким образом, можно сказать, что объект `Selection` открывает путь в документ, так как он предоставляет доступ к выделенному фрагменту документа. В частности, у объекта `Selection` имеется метод `TypeText(Text)`, с помощью которого можно вставлять текст в документ. Используя свойства этого объекта (которые, в свою очередь, могут являться объектами со своими свойствами), можно управлять параметрами выделенного фрагмента, например, устанавливать нужный размер и гарнитуру шрифта, выравнивать параграфы по центру и т.п.

Объектная модель Excel построена по тому же принципу, что и объектная модель Word. Основным объектом, содержащим все остальные, является `Application`. Напомним, что отдельные файлы в Excel называются *рабочими книгами*. Семейство `Workbooks` в Excel является аналогом семейства `Documents` в Word и содержит набор объектов `Workbook` (аналог объекта `Document` в Word), каждый из которых соответствует открытой в Word рабочей книге. Новая рабочая книга создается с помощью метода `Add()` объекта `Workbooks`.

Для доступа к ячейкам активного рабочего листа Excel используется свойство `Cells` объекта `Application`. Для получения или изменения значения отдельной ячейки применяется конструкция `Cells(row,column).Value`, где `row` и `column` являются соответственно номерами строки и столбца, на пересечении которых находится данная ячейка.

В Excel, как и в Word, имеется объект `Selection`, позволяющий работать с выделенным фрагментом электронной таблицы. Самым простым способом выделить диапазон ячеек активного рабочего листа является использование метода `Select()` объекта `Range`. Например, выражение `Range("A1:C1").Select()` позволяет выделить три смежные ячейки: `"A1"`, `"B1"` и `"C1"`.

Для того, чтобы понять, какой именно объект Word или Excel нужно использовать для решения той или иной задачи, часто проще всего бывает проделать в соответствующем приложении необходимые манипуляции вручную, включив предварительно режим записи макроса. В результате мы получим текст макроса на языке VBA (Visual Basic for Applications), из которого будет ясно, какие методы и с какими параметрами нужно вызывать и какие значения нужно присваивать свойствам объектов. В качестве простой иллюстрации проделаем следующие действия. Запустим Word, запустим Macro Recorder (**Сервис**|**Макрос**|**Начать запись…** (Tools|Macros|Record…)), назовем новый макрос "Пример1" и нажмем на кнопку "Ok". После этого напишем в документе слово "Пример" и прекратим запись макроса. Теперь можно посмотреть содержимое записанного макроса. Для этого нужно выбрать пункт **Макросы…** (Macroses…) в меню **Сервис**|**Макрос** (Tools|Macros), выделить макрос "Пример1" в списке всех доступных макросов и нажать кнопку **Изменить** (Edit). В открывшемся окне редактора Visual Basic появится текст макроса:

```
Sub Пример1()
'
' Пример1 Макрос
' Макрос записан 12.09.07 Андрей Владимирович Попов
'
    Selection.TypeText Text:="Пример"
End Sub
```

Как мы видим, для печати слова в документе был использован метод `TypeText` объекта `Selection`.

Макросы в Excel записываются и редактируются аналогичным образом.

Взаимодействие с Microsoft Word

Для того чтобы из сценария получить доступ к свойствам или методам внешнего сервера автоматизации, вначале надо "создать" соответствующий объект, т. е. загрузить в память экземпляр нужного COM-объекта и сохранить в переменной ссылку на этот объект. Объект в сценарии может создаваться несколькими способами:

- с помощью метода `CreateObject` объекта `WScript` (объектная модель WSH);
- с помощью конструкции `New ActiveXObject` (язык JScript);
- с помощью функции `CreateObject` (язык VBScript).

В любом случае в используемый метод или функцию в качестве параметра передается *программный идентификатор* объекта (ProgID), заключенный в скобки. Пример на языке VBScript:

```
Set WA=WScript.CreateObject("Word.Application")
```

Перед точкой в ProgID стоит имя *библиотеки типов* (type library) для объекта, которая может существовать как в виде отдельного файла с расширением tlb, так и в виде части файла с исполняемым кодом объекта (библиотека типов, содержащая сведения о COM-объекте, регистрируется в системном реестре при установке приложения, использующего этот объект). После точки в ProgID указывается имя класса, содержащего свойства и методы, доступные для использования другими приложениями.

Выполняя метод `CreateObject`, интерпретатор сценария через ProgID получает из системного реестра путь к файлам нужной библиотеки типов. Затем с помощью этой библиотеки в память загружается экземпляр запрашиваемого объекта, и его интерфейсы становятся доступными для использования в сценарии. Ссылка на созданный объект сохраняется в переменной; в дальнейшем, используя эту переменную, мы получаем доступ к свойствам и методам объекта, а также к его вложенным объектам (если они имеются).

Примеры управления приложением Microsoft Word из сценариев приведен в листинге 5.6. Сначала создается главный объект `Word.Application`, который запускает приложение Microsoft Word:

```
Set WA=WScript.CreateObject("Word.Application")
```

Затем создается новый пустой документ, в результате в переменную `WD` заносится ссылка на объект `Document`:

```
Set WD=WA.Documents.Add()
```

Наконец, в переменную `Sel` заносится ссылка на объект `Selection`, с помощью которого можно задать тип и размер шрифта, тип выравнивания абзацев и напечатать в документе строки текста:

```
Set Sel=WA.Selection;
```

В результате выполнения сценария PrintInWord.vbs в новом документе Microsoft Word печатаются две строки текста, после чего с помощью метода `PrintOut` объекта `Document` содержимое документа выводится на принтер:

```
WD.PrintOut();
```

Листинг 5.6. Использование сервера автоматизации Microsoft Word (VBScript)

```
'*****************************************************************
' Имя: PrintInWord.vbs
' Язык: VBScript
' Описание: Использование из сценария внешнего объекта
'           автоматизации (Microsoft Word)
'*****************************************************************
Option Explicit

Dim WA,WD,Sel  ' Объявляем переменные
'Создаем объект-приложение Microsoft Word
Set WA=WScript.CreateObject("Word.Application")
' Можно было использовать конструкцию
' Set WA=CreateObject("Word.Application")

Set WD=WA.Documents.Add  'Создаем новый документ (объект Document)
WA.Visible=true  ' Делаем Word видимым
Set Sel=WA.Selection  'Создаем объект Selection
Sel.Font.Size=14 'Устанавливаем размер шрифта
Sel.ParagraphFormat.Alignment=1  'Выравнивание по центру
Sel.Font.Bold=true  'Устанавливаем полужирный шрифт
Sel.TypeText "Привет!" & vbCrLf  'Печатаем строку текста
Sel.Font.Bold=false  'Отменяем полужирный шрифт
Sel.ParagraphFormat.Alignment=0  'Выравнивание по левому краю
'Печатаем строку текста
Sel.TypeText "Эти строки напечатаны с помощью WSH."
WD.PrintOut   'Выводим документ на принтер
'*************  Конец ********************************************
```

Взаимодействие с Microsoft Excel

В качестве примера управления сервером автоматизации Microsoft Excel рассмотрим сценарий PrintInExcel.vbs (листинг 5.7). Как и в случае с Microsoft Word, здесь сначала создается главный объект `Excel.Application`, который запускает приложение Microsoft Excel:

```
Set XL=WScript.CreateObject("Excel.Application")
```

Затем открывается новая рабочая книга:

```
XL.WorkBooks.Add
```

Ширина первого и второго столбца делается равной 40 пунктам, а третьей – 10 пунктам:

```
'Устанавливаем нужную ширину колонок
XL.Columns(1).ColumnWidth = 40
XL.Columns(2).ColumnWidth = 40
XL.Columns(3).ColumnWidth = 10
```

В столбцы первой строки записываются обозначения столбцов ("Фамилия", "Имя", "Телефон"), причем данный текст выделяется полужирным шрифтом:

```
'Печатаем в ячейки текст
XL.Cells(1,1).Value="Фамилия"
XL.Cells(1,2).Value="Имя"
XL.Cells(1,3).Value="Телефон"
'Выделяем три ячейки
XL.Range("A1:C1").Select
'Устанавливаем полужирный текст для выделенного диапазона
XL.Selection.Font.Bold = true
```

Во вторую строку заносятся данные для каждого столбца:

```
'Печатаем в ячейки текст
XL.Cells(2,1).Value="Иванов"
XL.Cells(2,2).Value="Иван"
XL.Cells(2,3).Value="555555"
```

Листинг 5.7. Использование сервера автоматизации Microsoft Excel (VBScript)

```
'*****************************************************************
' Имя: PrintInExcel.vbs
' Язык: VBScript
' Описание: Использование из сценария внешнего объекта
'           автоматизации (Microsoft Excel)
'*****************************************************************
Option Explicit

Dim XL  ' Объявляем переменные

'Создаем объект-приложение Microsoft Excel
Set XL=WScript.CreateObject("Excel.Application")
'Делаем окно Microsoft Excel видимым
XL.Visible=true
'Открываем новую рабочую книгу
XL.WorkBooks.Add
'Устанавливаем нужную ширину колонок
XL.Columns(1).ColumnWidth = 40
XL.Columns(2).ColumnWidth = 40
XL.Columns(3).ColumnWidth = 10
'Печатаем в ячейки текст
```

```
XL.Cells(1,1).Value="Фамилия"
XL.Cells(1,2).Value="Имя"
XL.Cells(1,3).Value="Телефон"
'Выделяем три ячейки
XL.Range("A1:C1").Select
'Устанавливаем полужирный текст для выделенного диапазона
XL.Selection.Font.Bold = true
'Печатаем в ячейки текст
XL.Cells(2,1).Value="Иванов"
XL.Cells(2,2).Value="Иван"
XL.Cells(2,3).Value="555555"
'*************  Конец ********************************************
```

Контрольные вопросы

Вопрос 1.

Вариант 1. С помощью каких методов можно запустить новый процесс в сценарии WSH?

a. метод Run объекта WshShell
b. метод Exec объекта WshShell
c. метод AppActivate объекта WshShell

Вариант 2. С помощью каких методов можно запустить новый независимый процесс в сценарии WSH?

a. метод Run объекта WshShell
b. метод Exec объекта WshShell
c. метод AppActivate объекта WshShell

Вопрос 2.

Вариант 1. С помощью какого метода можно завершить работу запущенного дочернего процесса?

a. метод с данной функциональностью в WSH отсутствует
b. метод Terminate объекта WshScriptExec
c. метод Kill объекта WshScriptExec

Вариант 2. Пусть objCalc – экземпляр объекта WshScriptExec, соответствующий запущенному дочернему процессу calc.exe (Калькулятор Windows). С помощью какой команды на языке VBScript можно завершить работу данного экземпляра Калькулятора?

a. objCalc.Terminate
b. objCalc.Quit
c. objCalc.Kill

Вопрос 3.

Вариант 1. С помощью какого метода можно имитировать нажатия клавиш в активном окне приложения Windows?

a. метод с данной функциональностью в WSH отсутствует
b. метод SendKeys объекта WshShell
c. метод PutKeys объекта WshShell

Вариант 2. С помощью какого метода можно имитировать нажатия клавиш в текущем сеансе командной строки?

a. имитировать нажатия клавиш можно только для приложений Windows

b. метод SendKeys объекта WshShell

c. метод PutKeys объекта WshShell

Вопрос 4

Вариант 1. Какой объект является корневым в объектной модели Microsoft Word?

a. Word

b. Application

c. Selection

Вариант 2. Какой программный идентификатор (ProgID) имеет корневой (главный) объект в объектной модели Microsoft Word?

a. Word.Application

b. Word.Server

c. Word.Selection

Лекция 6. Сценарии WSH как XML-документы. Схема WS XML

До сих пор мы рассматривали простые одиночные файлы сценариев, в которых мог использоваться язык JScript или VBScript. В версии WSH 1.0 это был единственный поддерживаемый тип сценариев, причем используемый язык определялся по расширению файла: js для JScript и vbs для VBScript. Начиная с WSH 2.0 появилась возможность создавать сценарии, в которых можно применять оба языка одновременно. Для таких сценариев в операционной системе регистрируется расширение wsf; wsf-файлы мы будем далее называть просто WS-файлами. Новый тип сценариев (WS-файл) имеет еще несколько важных преимуществ перед одиночными файлами сценариев WSH 1.0:

- поддерживаются вложенные файлы;
- возможен доступ из сценария к внешним мнемоническим константам, которые определены в библиотеках типов используемых объектов ActiveX;
- в одном WS-файле можно хранить несколько отдельных, независимых друг от друга, сценариев;
- сценарий становится *самодокументируемым*, т. е. вывод информации об использовании сценария и его синтаксисе происходит автоматически.

Понятно, что для обеспечения новых возможностей необходимо иметь больше информации, чем ее может предоставить отдельный сценарий. В самом файле сценария должна присутствовать некоторая дополнительная информация, скажем, имя этого сценария (подобная информация содержится, например, в заголовках HTML-страниц). Другими словами, для сценариев WSH должен использоваться уже некий специальный формат, а не просто отдельные js- или vbs-файлы. В качестве такого формата разработчики Microsoft выбрали язык XML (Extensible Markup Language), который уже использовался ими для определения информационной модели в технологии Windows Script Components (WSC), которая позволяет с помощью языков сценариев создавать и регистрировать полноценные COM-объекты.

Таким образом, теперь сценарии WSH не просто содержат в текстовом виде ActiveX-совместимый сценарий, а являются XML-приложениями, поддерживающими схему WS XML (Windows Script XML), которая, в свою очередь, опирается на схему WSC XML. Поэтому для понимания двух технологий (WSC и WSH) достаточно освоить одну схему XML.

WS-файл рассматривается сервером сценариев как файл с разметкой XML, который должен соответствовать схеме WS XML. Новый тип файла и формат XML обеспечивают более мощную среду для написания сценариев.

Для того чтобы использовать язык XML в сценариях WSH, вовсе не обязательно вникать во все тонкости этого языка, однако основные принципы XML понимать, конечно, нужно.

Основные принципы XML

Проявляемый в настоящее время большой интерес к языку XML объясняется тем, что он предоставляет возможности, позволяющие в текстовой форме описывать структурированные данные. Точнее говоря, XML является метаязыком для создания различных языков разметки, которые способны определять произвольные структуры данных — двоичные данные, записи в базе данных или сценарии. Прежде всего, XML используется в Internet-приложениях при работе браузеров, которые отображают информацию, находящуюся на Web-серверах. При этом пользователю отдельно передаются данные в виде XML-документа, и отдельно — правила интерпретации этих данных для отображения с помощью, например, языков сценариев JScript или VBScript.

Как и HTML, XML является независимым от платформы промышленным стандартом. Полные спецификации XML и связанных с ним языков доступны на официальной странице корпорации World Wide Web Consorctum (W3C) по адресу **http://www.w3c.org/xml**.

Внешне XML-документ похож на HTML-документ, так как XML-элементы также описываются с помощью *тегов*, то есть ключевых слов. Однако, в отличие от HTML, в XML пользователь может создавать собственные элементы, поэтому набор тегов не является заранее предопределенным. Еще раз повторим, что теги XML определяют структурированную информацию и, в отличие от тегов HTML, не влияют на то, как браузер отобразит эту информацию. Ниже перечислены несколько основных правил формирования корректного XML-документа:

- документ XML состоит из элементов разметки (markup) и непосредственно данных (content);
- все XML-элементы описываются с помощью тегов;
- в заголовке документа с помощью специальных тегов помещается дополнительная информация (используемый язык разметки, его версия и т. д.);
- каждый открывающий тег, который определяет область данных, должен иметь парный закрывающий тег (в HTML некоторые закрывающие теги можно опускать);
- в XML, в отличие от HTML, учитывается регистр символов;
- все значения атрибутов, используемых в определении тегов, должны быть заключены в кавычки;
- вложенность элементов в документе XML строго контролируется.

Рассмотрим теперь структуру и синтаксис WS-файлов, использующих схему WS XML.

Схема WS XML

Синтаксис элементов, составляющих структуру WS-файла, в общем виде можно представить следующим образом:

```
<element [attribute1="value1" [attribute2="value2" ... ]]>
   Содержимое (content)
</element>
```

Открывающий тег элемента состоит из следующих компонентов:

- открывающей угловой скобки "<";
- названия элемента, написанного строчными буквами;
- необязательного списка атрибутов со значениями (названия атрибутов пишутся строчными буквами, значения заключаются в двойные кавычки);
- закрывающей угловой скобки ">".

Например, тег начала элемента

```
<script language="JScript">
```

имеет имя тега `script` и определяет атрибут `language` со значением "JScript". Атрибуты предоставляют дополнительную информацию о соответствующем теге или последующем содержимом элемента. В нашем примере атрибут указывает на то, что содержимым элемента является текст сценария на языке JScript.

Закрывающий тег элемента состоит из следующих компонентов:

- открывающей угловой скобки "<";
- символа "/";
- названия элемента, написанного строчными буквами;
- закрывающей угловой скобки ">".

Таким образом, тег конца элемента не имеет атрибутов, например, `</script>`.

Если у элемента нет содержимого, то он имеет следующий вид:

```
<element [attribute1="value1" [attribute2="value2" … ]]/>
```

То есть в этом случае элемент состоит из следующих компонентов:

- открывающей угловой скобки "<";
- названия элемента, написанного строчными буквами;
- необязательного списка атрибутов со значениями (названия атрибутов пишутся строчными буквами, значения заключаются в двойные кавычки);
- символа "/";
- закрывающей угловой скобки ">".

Пример такого элемента:

```
<script language="JScript" src="tools.js"/>
```

Представленная в табл. 6.1 схема WS XML — это модель данных, определяющая элементы и соответствующие атрибуты, а также связи элементов друг с другом и возможную последовательность появления элементов. Также эта схема может задавать значения атрибутов по умолчанию.

Таблица 6.1. *Схема WS XML*

```
<?XML version="1.0" standalone="yes"?>
  <package>
    <job [id="JobID"]>
      <?job debug="true|false"?>
      <runtime>
        <named
        name="NamedName" helpstring="HelpString"
        type="string|boolean|simple" required="true|false"
        />
        <unnamed
        name="UnnamedName" helpstring="HelpString"
        many="true|false" required="true|false"
        />
        <description> Описание сценария </description>
        <example> Пример запуска сценария </example>
      </runtime>
      <resource id="ResourceID"> Строка или число </resource>
      <object id="ObjID"
      [classId="clsid:GUID"|progid="ProgID"]/>
      <reference
      [object="ProgID"|guid="typelibGUID"][version="version"]/>
      <script language="language" [src="strFileURL"]\>
      <script language="language" >
        <![CDATA[
        Код сценария
        ]]>
      </script>
    </job>
    Другие задания
  </package>
```

Таким образом, из табл. 6.1 видно, что:

- элемент `<package>` может содержать один или несколько элементов `<job>`;
- элемент `<job>` может содержать один или несколько элементов `<runtime>`, `<resource>`, `<object>`, `<reference>` или `<script>`;
- элемент `<runtime>` может содержать один или несколько элементов `<named>` и `<unnamed>`, а также элементы `<description>` и `<example>`.

Обязательными для создания корректного сценария являются только элементы <job> и <script>. Сам код сценария всегда располагается внутри элемента <script>.

Опишем теперь элементы XML, использующиеся в сценариях WSH, более подробно.

Элементы WS-файла

В WS-файл можно вставлять комментарии независимо от разметки XML. Сделать это можно двумя способами: с помощью элемента <!-- --> или элемента <comment>. Например:

```
<!-- Первый комментарий -->
```

или

```
<comment>
Второй комментарий
</comment>
```

Элементы *<?XML?>* и *<![CDATA[]]>*

Эти элементы являются стандартными для разметки W3C XML 1.0. В сценариях WSH они определяют способ обработки WS-файла. Всего существует два режима обработки сценария: нестрогий (loose) и строгий (strict).

При нестрогой обработке (элемент <?XML?> отсутствует) не предполагается выполнение всех требований стандарта XML. Например, не требуется различать строчные и заглавные буквы и заключать значения атрибутов в двойные кавычки. Кроме этого, в процессе нестрогой обработки считается, что все содержимое между тегами <script> и </script> является исходным кодом сценария. Однако при таком подходе может произойти ошибочная интерпретация вложенных в сценарий зарезервированных для XML символов или слов как разметки XML. Например, имеющиеся в коде сценария знаки "меньше" (<) и "больше" (>) могут привести к прекращению разбора и выполнения сценария.

Для того чтобы задать режим строгой обработки сценария, нужно поместить элемент <?XML?> в самой первой строке сценария — никаких других символов или пустых строк перед ним быть не должно. При такой обработке WS-файла нужно четко следовать всем правилам стандарта XML. Код сценария должен быть помещен в секцию CDATA, которая начинается с символов "<![CDATA[" и заканчивается символами "]]>".

Элемент *<?job?>*

Элемент <?job?> задает режим отладки при выполнении WS-файла. Если значение атрибута debug равно true, то задание может быть выполнено во внешнем отладчике. Если же значение атрибута debug равно false, то отладчик для этого задания применен быть не может. По умолчанию debug имеет значение false.

Элемент *<package>*

Этот элемент необходим в тех WS-файлах, в которых с помощью элементов <job> определено более одного задания. В этом случае все эти задания должны находиться внутри пары тегов <package> и </package> (см. табл. 6.1). Другими словами, <package> является контейнером для элементов <job>.

Если же в WS-файле определено только одно задание, то элемент <package> можно не использовать.

Элемент *<job>*

Элементы <job> позволяют определять несколько заданий (независимо выполняющихся частей) в одном WS-файле. Иначе говоря, между тегами <job> и </job> будет находиться отдельный сценарий (который, в свою очередь, может состоять из нескольких частей, написанных, возможно, на разных языках).

У элемента <job> имеется единственный атрибут id, который определяет уникальное имя задания. Например, в сценарии two_jobs.wsf определяются два задания с именами "Task1" и "Task2" (листинг 6.1).

Листинг 6.1. Файл two_jobs.wsf

```
<package>
<job id="Task1">
<!-- Описываем первое задание (id="Task1") -->
<script language="VBScript">
   WScript.Echo "Выполняется первое задание (VBScript)"
</script>
</job>
<job id="Task2">
<!-- Описываем второе задание (id="Task1") -->
<script language="JScript">
   WScript.Echo "Выполняется второе задание (JScript)"
</script>
</job>
</package>
```

Для того чтобы запустить конкретное задание из многозадачного WS-файла, нужно воспользоваться параметром //job:"JobID" в командной строке WSH. Например, следующая команда:

```
cscript //job:"Task1" two_jobs.wsf
```

запускает с помощью cscript.exe задание с именем "Task1" из файла two_jobs.wsf.

> ***Замечание***
>
> Если параметр //job не указан, то по умолчанию из многозадачного WS-файла запускается первое задание.

Если в WS-файле имеется несколько заданий, то они должны находиться внутри элемента <package>. Элемент <job> является одним из двух обязательных элементов в сценариях WSH с разметкой XML.

Элемент *<runtime>*

При запуске почти всех стандартных команд или утилит командной строки Windows с ключом /? на экран выводится встроенная справка, в которой кратко описываются назначение и синтаксис этой команды или утилиты. Хорошим тоном считается создание такой справки и для разрабатываемых сценариев WSH. Понятно, что добавление в сценарий функции вывода информации о назначении, синтаксисе и аргументах этого сценария потребовало бы написания довольно большого количества кода: необходимо следить за ключом /? в командной строке, а при добавлении нового параметра командной строки возникнет необходимость изменения функции, отвечающей за вывод информации на экран.

Элемент <runtime> позволяет сделать сценарий самодокументируемым, т. е. в этом случае при задании в командной строке ключа /? на экран будет автоматически выводиться информация об использовании сценарии, о его синтаксисе и аргументах (именных и безымянных), а также пример запуска сценария с конкретными значениями аргументов.

При этом сам элемент <runtime> является лишь контейнером, а содержимое для вывода информации хранится в элементы <named> (описание именных параметров командной строки), <unnamed> (описание безымянных параметров командной строки), <description> (описание самого сценария) и <example> (пример запуска сценария), которые находятся внутри <runtime>.

Элемент *<named>*

С помощью элементов <named> можно описывать (документировать) именные параметры командной строки сценария. В табл. 6.2 приведено описание аргументов элемента <named>.

Таблица 6.2. *Аргументы элемента <named>*

Аргумент	Описание
name	Задает имя параметра командной строки
helpstring	Строка, содержащая описание параметра командной строки
type	Определяет тип параметра командной строки. Может принимать значения "string" (символьный тип), "boolean" (логический тип), "simple" (в сценарий передается только имя параметра без дополнительного значения). По умолчанию используется тип "simple"
required	Используется для того, чтобы показать, является ли параметр командной строки обязательным. Может принимать значения "true" (параметр нужно указывать обязательно) и "false" (параметр можно не указывать)

Информация, которая указывается для объявляемого в элементе <named> параметра командной строки, используется только для самодокументируемости сценария и никак не влияет на реальные значения, которые будут указаны в командной строке при запуске сценария. Например, если параметр объявлен как обязательный (required="true"), но в действительности не был указан при запуске сценария, то никакой ошибки во время работы не произойдет.

Если для аргумента командной строки сценария указан тип "string", то предполагается, что этот аргумент имеет имя и значение, разделенные символом ":", например:

```
/Имя:"Андрей Попов" /Возраст:33
```

Если в качестве типа параметра командной строки используется "simple", то для этого параметра в командной строке указывается только его имя без значения:

```
/Имя /Возраст
```

Для того чтобы передать в сценарий аргумент командной строки типа "boolean", нужно после имени этого аргумента указать символ + (соответствует логическому значению "истина") или — (соответствует значению "ложь"). Например:

```
/Запись+ /ReWrite-
```

В листинге 6.2 приведен сценарий named.wsf, в котором в блоке <runtime> описываются три именных аргумента командной строки:

- /Имя (обязательный аргумент символьного типа);
- /Компьютер (необязательный аргумент символьного типа);
- /Новый (обязательный аргумент логического типа).

После запуска с помощью wscript.exe в сценарии named.wsf сначала вызывается метод WScript.Arguments.Usage, в результате чего на экран выводится диалоговое окно с информацией о сценарии и параметрах командной строки. Затем в сценарии проверяется, какие именно аргументы командной строки были подставлены при запуске и выделяются значения этих аргументов. Для этого создается объект WshNamed, являющийся коллекцией именных аргументов командной строки, и используется метод Exists этого объекта.

Значением параметра /Новый является константа логического типа (true или false), поэтому для формирования строки, соответствующей этому значению, используется условный оператор языка JScript:

```
//Проверяем, существует ли аргумент /Новый
if (objNamedArgs.Exists("Новый"))
  //Получаем с помощью условного оператора значение
  //логического аргумента /Новый
  s+="Новый пользователь: "+(objNamedArgs("Новый") ? "Да" : "Нет");
```

Листинг 6.2. Файл named.wsf

```
<job id="Named">
<runtime>
```

```
<description>
Имя: named.wsf
</description>
<named
   name="Имя"
   helpstring="Имя пользователя"
   type="string"
   required="true"
/>
<named
   name="Компьютер"
   helpstring="Имя рабочей станции"
   type="string"
   required="false"
/>
<named
   name="Новый"
   helpstring="Признак того, что такого пользователя раньше не было"
   type="boolean"
   required="true"
/>
</runtime>

<script language="JScript">
var objNamedArgs,s;
//Вызываем метод ShowUsage для вывода на экран описания сценария
WScript.Arguments.ShowUsage();
//Создаем объект WshNamed — коллекция именных аргументов сценария
objNamedArgs= WScript.Arguments.Named;
s="";
//Проверяем, существует ли аргумент /Имя:
if (objNamedArgs.Exists("Имя"))
  //Получаем значение символьного аргумента /Имя
  s+="Имя: "+objNamedArgs("Имя")+"\n";
//Проверяем, существует ли аргумент /Компьютер:
if (objNamedArgs.Exists("Компьютер"))
  //Получаем значение символьного аргумента /Компьютер
  s+="Машина: "+objNamedArgs("Компьютер")+"\n";
//Проверяем, существует ли аргумент /Новый
if (objNamedArgs.Exists("Новый"))
  //Получаем с помощью условного оператора значение
  //логического аргумента /Новый
  s+="Новый пользователь: "+(objNamedArgs("Новый") ? "Да" : "Нет");
//Выводим полученные строки на экран
WScript.Echo(s);
</script>
</job>
```

Элемент *<unnamed>*

С помощью элементов <unnamed> можно описывать (документировать) безымянные параметры командной строки сценария. В табл. 6.3 приведено описание аргументов элемента <unnamed>.

Таблица 6.3. *Аргументы элемента <unnamed>*

Аргумент	Описание
name	Задает имя, которое будет указано для описываемого параметра командной строки при выводе информации о сценарии
helpstring	Строка, содержащая описание параметра командной строки
many	Определяет, сколько раз может быть указан безымянный параметр в командной строке. Значение, равное "true" (используется по умолчанию), означает, что безымянный параметр может встретиться в командной строке более одного раза. Значение, равное "false", означает, что безымянный параметр должен быть указан только один раз
required	Определяет, является ли безымянный параметр командной строки обязательным. Может принимать значения "true", "on" или 1 (параметр нужно указывать обязательно), "false", "off" или 0 (параметр можно не указывать). Также значением аргумента "required" может быть целое число, которое показывает, сколько раз безымянный параметр должен обязательно быть указан в командной строке

Информация, которая указывается для объявляемого в элементе <unnamed> параметра командной строки, используется, как и в случае элемента <named>, только для самодокументируемости сценария и никак не влияет на реальные значения, которые будут указаны в командной строке при запуске сценария. Например, если безымянный параметр объявлен как обязательный (required="true"), но в действительности не был указан при запуске сценария, то никакой ошибки во время работы не произойдет.

Рассмотрим в качестве примера сценарий unnamed.wsf, в который в качестве параметров командной строки должны передаваться расширения файлов, причем обязательно должны быть указаны хотя бы два таких расширения (листинг 6.3).

Для создания информации об использовании этого сценарии создается элемент <unnamed> следующего вида:

```
<unnamed
   name="Расш"
   helpstring="Расширения файлов"
   many="true"
   required=2
/>
```

После запуска с помощью wscript.exe в сценарии unnamed.wsf сначала вызывается метод WScript.Arguments.Usage, в результате чего на экран выводится диалоговое окно с информацией о сценарии и параметрах командной строки. Затем в сценарии создается коллекция objUnnamedArgs (объект WshUnnamed), которая содержит все безымянные аргументы командной строки, реально переданные в сценарий:

```
objUnnamedArgs=WScript.Arguments.Unnamed;
```

После этого определяется общее число реально переданных в сценарий параметров командной строки (свойство `length`) и в цикле `while` организуется перебор всех элементов коллекции `objUnnamedArgs`.

Листинг 6.3. Файл unnamed.wsf

```
<job id="Unnamed">
<runtime>
<description>
Имя: unnamed.wsf
</description>
<unnamed
   name="Расш"
   helpstring="Расширения файлов"
   many="true"
   required=2
/>
</runtime>

<script language="JScript">
var objUnnamedArgs,s;
//Вызываем метод ShowUsage для вывода на экран описания сценария
WScript.Arguments.ShowUsage();
objUnnamedArgs=WScript.Arguments.Unnamed;  //Создаем объект
WshUnnamed
//Определяем количество безымянных аргументов
s="Передано в сценарий безымянных аргументов:
"+objUnnamedArgs.length;
for (i=0; i<=objUnnamedArgs.length-1; i++)
  //Формируем строки со значениями безымянных аргументов
  s+="\n"+objUnnamedArgs(i);
//Выводим полученные строки на экран
WScript.Echo(s);
</script>
</job>
```

Элемент *<description>*

Внутри элемента `<description>` помещается текст (без дополнительных кавычек), описывающий назначение сценария. Как и все элементы внутри `<runtime>`, этот текст выводится на экран, если сценарий был запущен с ключом `/?` в командной строке или если в сценарии встретился вызов метода `ShowUsage` объекта `WshArguments`. При выводе текста на экран учитываются все имеющиеся в нем пробелы, символы табуляции и перевода строки.

Пример использования элемента `<description>` и метода `ShowUsage` представлен в сценарии descrip.wsf (листинг 6.4). Здесь сразу вызывается метод `WScript.Arguments.ShowUsage`, в результате чего на экран выводится

диалоговое окно (в случае запуска сценария с помощью wscript.exe) или просто строки текста (в случае запуска сценария с помощью cscript.exe) с описанием запущенного сценария.

Листинг 6.4. Файл descrip.wsf

```
<job id="Descrip">
<runtime>
<description>
Имя: descrip.wsf
Описание: Здесь можно привести дополнительное описание сценария
</description>
</runtime>
<script language="JScript">
//Вызываем метод ShowUsage
WScript.Arguments.ShowUsage();
</script>
</job>
```

Элемент *<example>*

Внутри элемента `<example>` приводится текст из одной или нескольких строк, в котором можно описать примеры запуска сценария. Если сценарий был запущен с ключом `/?` в командной строке или в сценарии встретился вызов метода `ShowUsage` объекта `WshArguments`, то этот текст выводится в графическое диалоговое окно (при использовании wscript.exe) или на экран (в консольном режиме при использовании cscript.exe). При выводе текста на экран учитываются все имеющиеся в нем пробелы, символы табуляции и перевода строки, при этом строки из элемента `<example>` выводятся после строк из элемента `<description>`.

Пример использования элемента `<example>` приведен в листинге 6.5.

Листинг 6.5. Файл example.wsf

```
<job id="Example">
<runtime>
<description>
Имя: example.wsf
Описание: Здесь можно привести дополнительное описание сценария
</description>
<example>

Здесь приводится пример запуска сценария
(с параметрами командной строки, например)
</example>
</runtime>
<script language="JScript">
//Вызываем метод ShowUsage
WScript.Arguments.ShowUsage();
</script>
```

```
</job>
```

Элемент *<resource>*

Элемент `<resource>` позволяет отделить символьные или числовые константы (ресурсы) от остального кода сценария. Например, таким образом удобно собрать в одном месте строки, которые используются в сценарии для вывода каких-либо стандартных сообщений. Если после этого понадобится изменить сообщения в сценарии (например, перевести их на другой язык), то достаточно будет внести соответствующие корректировки в строки, описанные в элементах `<resource>`.

Для получения значения ресурса в сценарии нужно вызвать метод `getResource`, передав в качестве параметра символьный идентификатор ресурса (значение атрибута `id`).

В листинге 6.6 представлен пример сценария resource.wsf, в котором определяется ресурсная строка с идентификатором `"MyName"`:

```
<resource id="MyName">
Меня зовут Андрей Попов
</resource>
```

Значение этого ресурса затем выводится на экран с помощью метода `Echo` объекта `WScript` и метода `getResource`:

```
WScript.Echo(getResource("MyName"));
```

Листинг 6.6. Файл resource.wsf

```
<job id="Resource">
<runtime>
<description>
Имя: resource.wsf
Описание: Пример использования в сценарии ресурсных строк
</description>
</runtime>
<resource id="MyName">
Меня зовут Андрей Попов
</resource>
<script language="JScript">
//Выводим на экран значение ресурса "MyName"
WScript.Echo(getResource("MyName"));
</script>
</job>
```

Элемент *<object>*

Элемент `<object>` предлагает еще один способ создания экземпляра COM-объектов для использования их внутри сценариев. Напомним, что ранее для этого мы использовали методы `CreateObject` и `GetObject` объекта `WScript`, объект `ActiveXObject` и функцию `GetObject` языка JScript, а также функцию `CreateObject` языка VBScript. Элемент `<object>` может заменить эти средства.

Атрибут `id` в `<object>` — это имя, применяемое для обращения к объекту внутри сценария. Отметим, что объект, создаваемый с помощью тега `<object>`, будет глобальным по отношению к тому заданию, в котором он определен. Другими словами, этот объект может использоваться во всех элементах `<script>`, находящихся внутри элемента `<job>`, содержащего описание объекта.

Атрибуты `classid` и `progid` используются в `<object>` соответственно для указания *глобального кода* создаваемого объекта (Globally Unique ID, GUID) или *программного кода* объекта (Programmic Identifier). Из этих двух необязательных атрибутов может быть указан только один. Например, создать объект `FileSystemObject` (`GUID="0D43FE01-F093-11CF-8940-00A0C9054228"`) можно двумя способами:

```
<object id="fso" classid="clsid:0D43FE01-F093-11CF-8940-
00A0C9054228"/>
```

или

```
<object id="fso" progid="Scripting.FileSystemObject"/>
```

В обычном js-файле или внутри элемента `<script>` этот объект мы бы создали следующим образом:

```
var fso = WScript.CreateObject("Scripting.FileSystemObject");
```

или

```
var fso = new ActiveXObject("Scripting.FileSystemObject");
```

Элемент *<reference>*

При вызове многих методов внешних объектов, которые используются внутри сценариев, требуется указывать различные числовые или строковые константы, определенные в этих внешних объектах. Например, для того, чтобы открыть текстовый файл с помощью метода `OpenTextFile` объекта `FileSystemObject`, может потребоваться указать параметр, который определяет режим ввода/вывода (возможные значения констант `ForReading=1`, `ForWriting=2` и `ForAppending=8`) открываемого файла. Ясно, что запомнить все значения констант различных объектов очень трудно, поэтому при их использовании приходится постоянно обращаться к справочной информации.

К счастью, большинство объектов предоставляет информацию об именах используемых ими констант в своей библиотеке типов, которая регистрируется в системном реестре при установке COM-объекта и может существовать как в виде отдельного файла с расширением tlb, так и в виде части файла с исполняемым кодом объекта. Элемент `<reference>` как раз обеспечивает доступ к мнемоническим константам, определенным в библиотеке типов объекта (экземпляр объекта при этом не создается).

Для того чтобы воспользоваться константами определенного объекта, нужно в теге `<reference>` указать программный код этого объекта (атрибут `object`) или *глобальный код* его библиотеки типов (атрибут `guid`), а также, при необходимости, номер версии объекта (атрибут `version`).

Например, доступ к константам объекта `FileSystemObject` организуется следующим образом:

```
<reference object="Scripting.FileSystemObject"/>
```

После этого в сценариях можно просто использовать константы с именами `ForReading` или `ForAppending`, не заботясь об их числовых значениях.

Элемент <*script*>

Элемент `<script>` с помощью атрибута `language` позволяет определить язык сценария (`language="JScript"` для языка JScript и `language="VBScript"` для языка VBScript). Это делает возможным использовать в одном задании сценарии, написанные на разных языках (*мультиязычные сценарии*), что иногда бывает очень удобно. Предположим, что у вас имеются сценарии на JScript и VBScript, функции которых необходимо объединить. Для этого не нужно переписывать один из сценариев на другой язык — используя WS-файл, можно из сценария JScript спокойно вызывать функции, написанные на VBScript и наоборот.

Атрибут `src` позволяет подключить к выполняющемуся сценарию внешний файл с другим сценарием. Например, задание элемента

```
<script language="JScript" src="tools.js"/>
```

приведет к такому же результату, как если бы содержимое файла tools.js было расположено между тегами `<script>` и `</script>`:

```
<script language="JScript">
   Содержимое файла tools.js
</script>
```

Таким образом, можно выделить код, который должен использоваться в нескольких сценариях, поместить его в один или несколько внешних файлов, а затем по мере необходимости просто подключать с помощью атрибута `src` эти файлы к другим сценариям.

Примеры сценариев с разметкой XML

Приведем примеры сценариев, иллюстрирующие основные свойства WS-файлов.

Несколько заданий в одном файле

Каждое отдельное задание в WS-файле должно находиться внутри элементов `<job>` и `</job>`. В свою очередь, все элементы `<job>` являются дочерними элементами контейнера `<package>`.

В качестве примера рассмотрим сценарий multijob.wsf, приведенный в листинге 6.7. Здесь описываются два задания с идентификаторам "VBS" (сценарий на языке VBScript) и "JS" (сценарий на языке JScript).

Листинг 6.7. Файл multijob.wsf

```
<package>
<job id="VBS">
<!-- Описываем первое задание (id="VBS") -->
```

```
<runtime>
<description>
Имя: multijob.wsf
Описание: Первое задание из multijob.wsf
</description>
</runtime>
<script language="VBScript">
   WScript.Echo "Первое задание (VBScript)"
</script>
</job>
<job id="JS">
<!-- Описываем второе задание (id="JS") -->
<description>
Имя: multijob.wsf
Описание: Второе задание из multijob.wsf
</description>
</runtime>
<script language="JScript">
   WScript.Echo("Второе задание (JScript)");
</script>
</job>
</package>
```

Для того чтобы выполнить первое задание сценария multijob.wsf, которое выведет на экран строку "Первое задание (VBScript)", нужно выполнить одну из следующих команд:

```
cscript //job:"VBS" multijob.wsfsf
cscript multijob.wsf
wscript //job:"VBS" multijob.wsf
wscript multijob.wsf
```

Для запуска второго задания, выводящего на экран строку "Второе задание (JScript)", нужно явно указывать идентификатор этого задания, поэтому используется одна из двух команд:

```
cscript //job:"JS" multijob.wsf
wscript //job:"JS" multijob.wsf
```

Использование констант внешних объектов

Для того чтобы в сценарии обращаться по имени к константам, определенным во внешних объектах, не создавая экземпляров самих объектов, необходимо сначала получить ссылку на эти объекты с помощью элемента `<reference>`.

В листинге 6.8 приведен сценарий refer.wsf, в котором с помощью элемента `<reference>` производится доступ к трем константам объекта `FileSystemObject` (ForReading, ForWriting и ForAppending), которые определяют режим работы из сценария с внешним текстовым файлом. В результате выполнения сценария refer.wsf на экран выведется диалоговое окно с информацией о значениях констант объекта `FileSystemObject`.

Листинг 6.8. Использование в сценарии констант внешних объектов (файл refer.wsf)

```
<job id="Example">
<runtime>
<description>
Имя: refer.wsf
Описание: Использование констант внешних объектов
</description>
</runtime>
<!-- Получаем ссылку на объект FileSystemObject -->
<reference object="Scripting.FileSystemObject"/>
<script language="JScript">
var s;
  s="Значения констант объекта FileSystemObject:\n\n";
  //Получаем значение константы ForReading
  s+="ForReading="+ForReading+"\n";
  //Получаем значение константы ForWriting
  s+="ForWriting="+ForWriting+"\n";
  //Получаем значение константы ForAppending
  s+="ForAppending="+ForAppending;
  //Выводим полученные строки на экран
  WScript.Echo(s);
</script>
</job>
```

Подключение внешних файлов

К WS-файлу можно подключать "обычные" JScript- или VBScript-сценарии, которые находятся во внешних файлах. Для этого нужно указать путь к этому внешнему файлу в атрибуте `src` элемента `<script>`.

Для примера создадим файл inc.js, содержащий строку

```
WScript.Echo("Здесь выполняется сценарий inc.js");
```

и файл main.wsf, содержание которого приведено в листинге 6.9.

Листинг 6.9. Подключение внешнего сценария (файл main.wsf)

```
<job id="Example">
<runtime>
<description>
Имя: main.wsf
Описание: Подключение сценария, находящегося во внешнем файле
</description>
</runtime>
<!-- Подключаем сценарий из файла inc.js -->
<script language="JScript" src="inc.js"/>
<!-- Определяем основной сценарий -->
<script language="JScript">
  WScript.Echo("Здесь выполняется основной сценарий");
</script>
</job>
```

Если запустить main.wsf с помощью cscript.exe, то на экран выведутся две строки:

```
Здесь выполняется сценарий inc.js
Здесь выполняется основной сценарий
```

Различные языки внутри одного задания

Ни в WSH, ни в JScript нет метода или функции, которые позволяли бы в графическом режиме создать диалоговое окно для ввода текста. Однако в языке VBScript имеется функция `InputBox`, предназначенная как раз для этой цели; используя разметку XML, мы можем легко использовать эту функцию в сценариях JScript. Соответствующий пример приведен в сценарии multilang.wsf (листинг 6.10).

Сначала в этом сценарии на языке VBScript описывается функция `InputName`, которая возвращает строку, введенную с помощью функции `InputBox`:

```
<script language="VBScript">
  Function InputName
  InputName = InputBox("Введите Ваше имя:", "Окно ввода VBScript")
  End Function
</script>
```

Затем в следующем разделе `<script>` приводится JScript-сценарий, в котором происходит вызов функции `InputName` и сохранение возвращаемого ею значения в переменной s:

```
var s;
    s = InputName();
```

Значение полученной таким образом переменной `s` выводится затем на экран:

```
WScript.Echo("Здравствуйте, "+s+"!");
```

Листинг 6.10 Использование различных языков внутри одного задания (файл multilang.wsf)

```
<job id="Example">
<runtime>
<description>
Имя: multilang.wsf
Описание: Использование функции InputBox в JScript-сценарии
</description>
</runtime>
<script language="VBScript">
  Function InputName  ' Описываем функцию на языке VBScript
  ' Вводим имя в диалоговом окне
  InputName = InputBox("Введите Ваше имя:", "Окно ввода VBScript")
  End Function
</script>
<script language="JScript">
    var s;
    s = InputName();  //Вызываем функцию InputName
```

```
    //Выводим значение переменной s на экран
    WScript.Echo("Здравствуйте, "+s+"!");
</script>
</job>
```

Контрольные вопросы

Задача 1.

Вариант 1. Какие из указанных ниже элементов входят в схему WS XML?

a. <runtime>
b. <reference>
c. <object>

Вариант 2. Какие из указанных ниже элементов входят в схему WS XML?

a. <script>
b. <header>
c. <data>

Задача 2.

Вариант 1. В каких случаях в сценарии необходимо использовать элемент <package>?

a. сценарий содержит код на нескольких языках
b. сценарий содержит более одного задания
c. элемент <package> необходимо указывать во всех сценариях

Вариант 2. В каких случаях в сценарии может отсутствовать элемент <package>?

a. сценарий содержит единственное задание
b. в сценарии описаны несколько заданий, но все они написаны на одном языке
c. элемент <package> необходимо указывать во всех сценариях

Задача 3.

Вариант 1. Какие из приведенных ниже элементов могут содержаться внутри элемента <runtime> в схеме WS XML?

a. <named>
b. <unnamed>
c. <arguments>

Вариант 2. Какие теги могут использоваться для документирования сценария с разметкой XML?

a. <description>
b. <info>
c. <example>

Задача 4.

Вариант 1. Можно ли в сценарии с разметкой XML использовать константы, определенные во внешних COM-объектах, обращаясь к ним по имени?

a. нельзя, именованные константы необходимо определять в сценарии явно

b. можно, предварительно установив связь с библиотекой типов нужных объектов с помощью элемента <reference>

c. можно, предварительно подключив нужный объект с помощью элемента <object>

Вариант 2. Какой тег должен присутствовать в сценарии с разметкой XML, чтобы из этого сценария можно было пользоваться константами объекта FileSystemObject без их предварительного объявления в сценарии?

a. <reference object="Scripting.FileSystemObject"/>

b. <object id="fso" progid="Scripting.FileSystemObject"/>

c. из сценария с разметкой XML нельзя пользоваться константами внешних объектов

Практические задания

Вариант 1

1. Написать сценарий `changea.vbs`, который переключал бы атрибут "Архивный" у файла, имя которого задается в качестве параметра командной строки. Например, если у файла example.txt был установлен атрибут "Архивный", то команда

   ```
   cscript changea.js example.txt
   ```

   должна сбросить этот атрибут у example.txt; если же первоначально файл не имел атрибута "Архивный", то его нужно установить. В сценарии должна быть предусмотрена проверка наличия указанного файла на диске.
2. Создать с помощью сценария текстовый файл 'Отчет по дискам.txt', в котором содержалась бы информация о дате и времени создания отчета, а также таблица использования дискового пространства на всех логических дисках локального компьютера (буква диска, метка тома, общий объем диска, объем свободной и используемой частей диска). Полученный файл с отчетом автоматически открыть с помощью Блокнота Windows.
3. Написать сценарий, сохраняющий информацию о логических дисках компьютера (см. задание 2), в файл Microsoft Word.
4. Вывести с помощью сценария имена всех файлов, находящихся на рабочем столе активного пользователя.
5. Оформить сценарии заданий 1–4 в виде одного многозадачного сценария Variant1.wsf.

Вариант 2

1. Написать сценарий `changea.vbs`, который переключал бы атрибут "Только для чтения" у файла, имя которого задается в качестве параметра командной строки. Например, если у файла example.txt был установлен атрибут "Только для чтения", то команда

   ```
   cscript changea.js example.txt
   ```

   должна сбросить этот атрибут у example.txt; если же первоначально файл не имел атрибута "Только для чтения", то его нужно установить. В сценарии должна быть предусмотрена проверка наличия указанного файла на диске.
2. Создать с помощью сценария HTML-файл 'Отчет по дискам.htm', в котором содержалась бы информация о дате и времени создания отчета, а также таблица использования дискового пространства на всех жестких дисках локального компьютера (буква диска, метка тома, общий объем диска, объем свободной и используемой частей диска). Полученный файл с отчетом автоматически открыть с помощью браузера Internet Explorer.
3. Написать сценарий, сохраняющий информацию о логических дисках компьютера (см. задание 2), в файл Microsoft Excel.

4. Вывести с помощью сценария имена всех файлов, находящихся в папке "Избранное" активного пользователя.
5. Оформить сценарии заданий 1–4 в виде одного многозадачного сценария Variant2.wsf.

Ответы к контрольным вопросам

Лекция 1

Вопрос	Вариант 1	Вариант 2
1	a	b
2	a, b	c
3	c	c

Лекция 2

Вопрос	Вариант 1	Вариант 2
1	a	b
2	a	b
3	a	b
4	b	c

Лекция 3

Вопрос	Вариант 1	Вариант 2
1	c	a
2	a	b
3	b	a
4	a, b, c	a

Лекция 4

Вопрос	Вариант 1	Вариант 2
1	c	a, b
2	b	b
3	b	a
4	b	a

Лекция 5

Вопрос	Вариант 1	Вариант 2
1	a, b	a
2	b	a
3	b	a
4	b	a

Лекция 6

Вопрос	Вариант 1	Вариант 2
1	a, b	a
2	b	a
3	a, b	a, c
4	b	a

Библиографический список

1. Борн Г. Руководство разработчика на Microsoft Windows Script Host 2.0. Мастер-класс: Пер. с англ. / Г. Борн – СПб.: Питер; М.: Изд.-торг. дом "Русская редакция", 2001. – 480 с.
2. Коробко И. Администрирование сетей Windows с помощью сценариев / И. Коробко. – СПб.: БХВ-Петербург, 2007. – 368 с.
3. Попов А.В. Командные файлы и сценарии Windows Script Host / А.В. Попов. – СПб.: БХВ-Петербург, 2002. – 320 с.
4. Попов А.В. Windows Script Host для Windows 2000/XP / А.В. Попов – СПб.: БХВ-Петербург, 2003. – 640 с.

Содержание