ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Федеральное государственное образовательное бюджетное учреждение высшего профессионального образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ им. проф. М. А. БОНЧ-БРУЕВИЧА»

А. Н. Губин
Ф. В. Филиппов

ОСНОВЫ ПРОГРАММИРОВАНИЯ СЦЕНАРИЕВ В СРЕДЕ WSH

УЧЕБНОЕ ПОСОБИЕ

СПбГУТ)))

САНКТ-ПЕТЕРБУРГ
2013

УДК 004.43(075.8)
ББК 32.973-018.1я73
Г93

Рецензент
кандидат технических наук, доцент кафедры робототехники
и автоматизации производственных систем СПбГЭТУ «ЛЭТИ»
А. В. Шевченко

*Утверждено редакционно-издательским советом СПбГУТ
в качестве учебного пособия*

Губин, А. Н.

Г93 Основы программирования сценариев в среде WSH : учебное пособие / А. Н. Губин, Ф. В. Филиппов ; СПбГУТ. – СПб., 2013. – 96 с.

Рассматриваются основные принципы программирования сценариев на языке VBScript в среде Windows Script Host и их использования для управления приложениями и администрирования компьютерных систем и сетей.

Пособие предназначено для бакалавров и магистров специальности 230400 «Информационные системы и технологии», а также аспирантов и специалистов в области информационных технологий.

УДК 004.43(075.8)
ББК 32.973-018.1я73

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Основными элементами современных информационных систем (ИС) как правило, являются вычислительные структуры. Реализация алгоритмов функционирования этих структур в большинстве своем подразумевает их программную реализацию. Достаточно часто в качестве вычислительных элементов ИС используются компьютеры под управлением операционных систем (ОС) семейства Windows. Такие компьютеры используются для реализации функций автоматизированных рабочих мест в различных ИС. В процессе разработки и проектирования подобных систем возникает необходимость решения задач не только обработки информации на прикладном уровне, но и на канальном и сетевом уровнях, а также управления режимами работы самих компьютеров.

До операционной системы Windows-98 в ОС Windows практически не было средств автоматизации выполнения рутинных операций обработки служебной информации и управления режимами работы компьютера. Определенные задачи, такие как копирование файлов и изменение некоторых параметров ОС можно было выполнить с помощью командных файлов MS-DOS (BAT-файлов). Однако в BAT-файлы можно помещать лишь простые последовательности команд MS-DOS с достаточно серьезными ограничениями по выполняемым функциям (нет поддержки диалоговых окон, ограничения поддержки циклов и др.).

Результатом развития средств автоматизации управления ОС и обработки информации явилось появление в списке технологий Microsoft достаточно мощного средства – технологии Microsoft Windows Script, которая реализована автономным сервером сценариев *Windows Script Host* (WSH).

WSH позволяет исполнять сценарии на уровне ОС. Можно вызвать исполнение сценария из командной строки или из Windows Explorer, дважды щелкнув по выделенному имени файла сценария.

С самого начала для обеспечения обмена данными между приложениями в ОС Windows использовалась технология связывания и внедрения объектов (OLE – *Object Linking and Embedding*). Вначале технология OLE использовалась только для создания составных документов, а затем и для решения задач предоставления приложениям функций других приложений. Технология, позволяющая одному приложению вызывать функции другого приложения, получила название OLE Automation. В основе OLE и OLE Automation лежит разработанная Microsoft базовая компонентная технология COM (Component Object Model).

Компонентное программное обеспечение – это способ разработки программ, при котором используется технологии, подобные технологиям 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-то языке программирования, детали реализации используемых алгоритмов также скрыты внутри компонента, разработчику доступны только интерфейсы, которыми могут пользоваться и другие приложения.

В настоящее время, вместо термина OLE используется термин ActiveX, обозначающий компоненты (объекты), созданные на базе COM технологии.

Технология ActiveX до последнего времени являлась ключевой в продуктах Microsoft (Microsoft Office, Internet Explorer и др.). В эти продукты для управления объектами были встроены интерпретаторы специальных языков сценариев: VBScript и JScript, однако вне этих продуктов выполнять сценарии, написанные на VBScript и JScript, было нельзя.

Сервер сценариев WSH устраняет этот недостаток, предоставляя единый интерфейс для языков сценариев, которые позволяют использовать любые внешние объекты ActivX. С помощью WSH сценарии могут быть исполнены непосредственно в ОС Windows, без встраивания их в HTML-страницы или другие объекты Microsoft Office.

В пособии рассмотрены основные вопросы создания и использования сценариев на VBScript, а в приложении приведен краткий справочник по языку программирования сценариев VBScript.

Ограниченный объем пособия не позволяет достаточно подробно осветить все вопросы, касающиеся всех важнейших проблем программирования сценариев на VBScript и JScript в среде WSH, поэтому для получения дополнительных сведений рекомендуется обратиться к соответствующим источникам, список которых включает 3 наименования. При работе над пособием авторы неоднократно осуществляли заимствование из этих источников определений, примеров и методов изложения, без отдельных ссылок в тексте.

В заключение авторы благодарят рецензентов и редакторов за внимательное прочтение рукописи и замечания, способствовавшие улучшению качества предлагаемого пособия.

1. СОЗДАНИЕ И ЗАПУСК ПРОСТЕЙШИХ СЦЕНАРИЕВ

1.1. Серверы сценариев

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 VBScript-сценарий, выводящий на экран диалоговое окно с надписью «Привет Всем!». Для этого достаточно с помощью, например стандартного Блокнота Windows (notepad.exe) создать файл Privet.vbs, содержащий всего одну строку:

```
WScript.Echo «Привет Всем!».
```

Для обеспечения работы этого сценария достаточно всего одной строки, однако полезно добавить в начало файла информацию о находящемся в нем сценарии: имя файла, используемый язык, краткое описание выполняемых действий. Для нашего сценария можно использовать следующий вариант оформления содержимого файла (рис. 1.1).

```
‘========================
‘Язык – VBScript
’Имя файла – Privet.vbs
‘Автор – Иванов И.И.
‘ Дата – 12.02.2013
‘Вывод на экран
‘========================
WScript.Echo “Привет всем!”
```

Рис. 1.1. Пример оформления листинга файла-сценария

Запустить сценарий в работу можно несколькими способами.

Исполняемые файлы сервера сценариев расположены в директории %WINDIR%system32 и содержат два основных файла. Файл WScript.exe – оконно-диалоговый сервер сценариев и файл CScript.exe – консольный сервер сценариев. Разница в работе этих серверов состоит только в способе и форме вывода информации при выполнения скрипта – WScript генерирует вывод, основанный на диалоговых окнах, в то время как CScript возвращает вывод в командное окно, в котором был запущен скрипт.

Запустим сценарий, записанный в файле Privet.vbs из командной строки. Для этого необходимо запустить режим командной строки и выполнить в нем команду:

Cscript F:\PRIMER_WSH\Privet.vbs ,

где F:\PRIMER_WSH\Privet.vbs – полное (с указанием пути к файлу) имя файла-сценария. Результат работы сценария представлен на рис. 1.2.

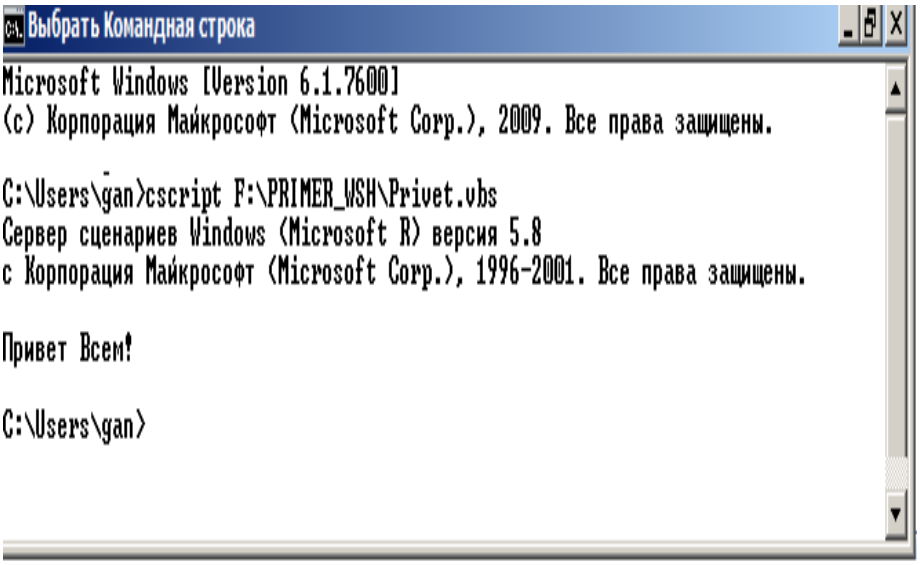


Рис. 1.2. Выполнение сценария Privet.vbs в консольном режиме

Этот сценарий можно выполнить из командной строки с помощью графической или оконно-диалоговой версии сервера сценариев.

Для этого необходимо в режиме командной строки выполнить команду:

WCscript F:\PRIMER_WSH\Privet.vbs.

При выполнении этой команды на экране появится диалоговое окно (рис. 1.3).

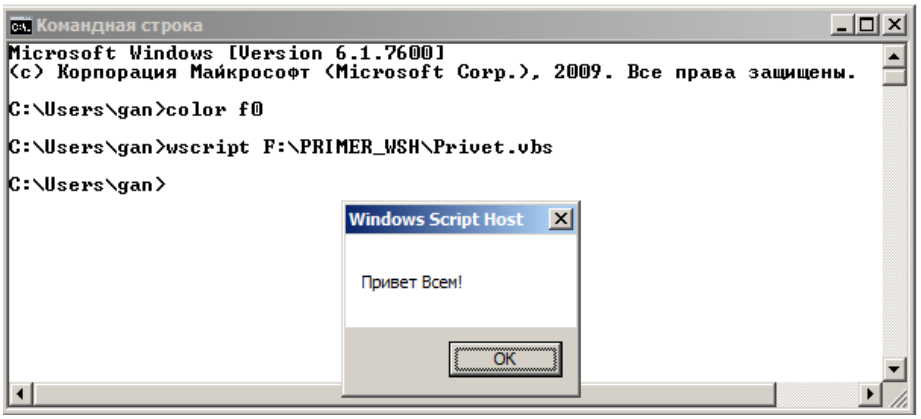


Рис. 1.3. Выполнение сценария Privet.vbs из командной строки с использованием графической версии сервера сценариев

Запустить сценарий на исполнение можно с помощью пункта Выполнить меню – Пуск. Для этого необходимо набрать полное имя файла-сценария в поле Открыть (для Windows XP) или в поле Найти программы и файлы (для Windows 7). В этом случае по умолчанию сценарий будет выполнен с помощью сервера Wscript.exe. Результат работы файла будет представлен в виде графического диалогового окна.

Проще всего запустить сценарий, используя окно Проводника Windows или Рабочего стола. В этом случае достаточно выполнить двойной щелчок мышью на имени файла или на его значке.

Вопросы для самопроверки

1. Как запустить в работу консольный сервер сценариев WSH?
2. С помощью запуска какого файла реализуется оконно-диалоговый режим работы сервера сценариев WSH?
3. Назовите простейшее средство для создания сценариев WSH.
4. Назовите основные способы запуска сценариев WSH.

1.2. Передача параметров серверу сценариев

В общем виде запуск сценариев через командную строку реализуется в следующей форме:

```
WScript имя сценария.расширение [параметры сервера] [аргументы сценария]
    или
CScript имя сценария.расширение [параметры сервер] [аргументы сценария]
```

Перед параметрами сервера, которые включают или выключают различные опции Windows Script Host, ставятся два символа прямого слэша (//).

Аргументы скрипту можно передавать, указывая их после параметров (если они используются), отделяя от параметров пробелом.

Полный список параметров сервера позволяет вывести команда CScript //? (рис. 1.4).

Список параметров для WScript.exe можно получить, если в меню «Пуск» выбрать команду «Выполнить», в окне которой набрать WScript //?.

Параметры исполнения сценария, запускаемого из командной строки, задаются параметрами сервера, указываемые в командной строке.

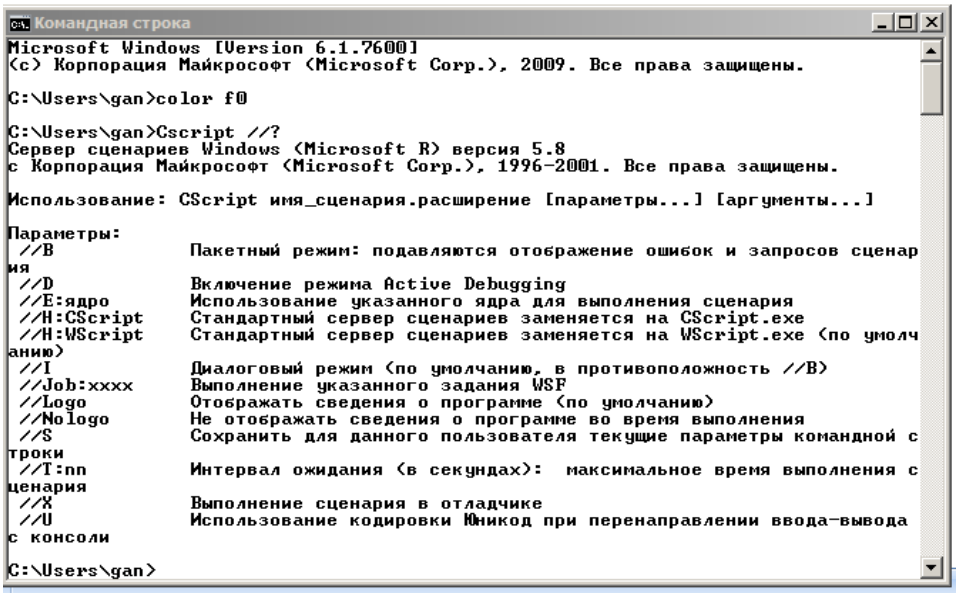


Рис. 1.4. Результат выполнения команды

Так, исполнение сценария Prvet.vbs с параметром //nologo приводит к результату (рис. 1.5), который отличается от результата исполнения этого сценария без параметров (рис. 1.2) – отсутствует информация о версии WSH.

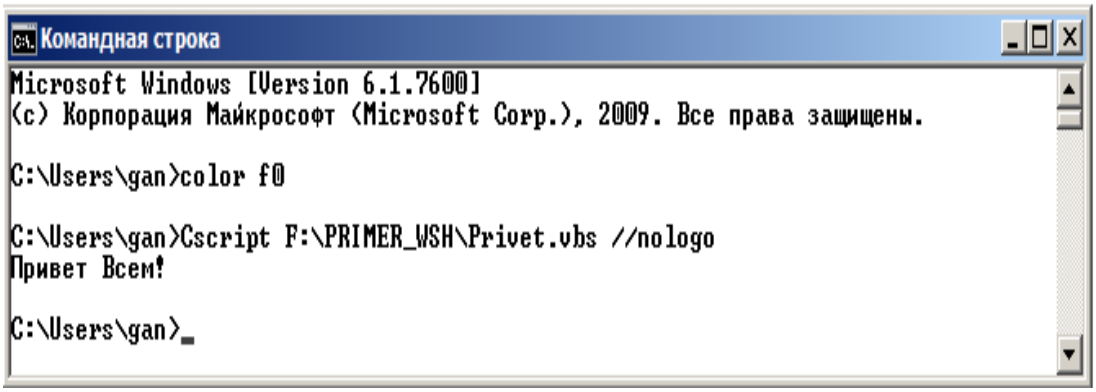


Рис. 1.5. Выполнение сценария Privet.vbs с параметром nologo.

Если сценарий запускается с использованием графического интерфейса (из Windows), то задать параметры сервера можно из диалогового окна «Выполнить».

В этом случае окно с результатом работы скрипта погаснет через 5 с после появления.

При запуске сценария двойным щелчком по значку сценария (или по его ярлыку) параметры сервера можно определить через дополнительный файл с именем сценария и расширением wsh.

Для этого необходимо щелкнуть по файлу сценария правой кнопкой, выбрать из контекстного меню пункт Properties (Свойства) и в открывшемся окне свойств сценария установить нужные значения (рис. 1.6). Окно за-

кроется, если щелкнуть по кнопкам «Применить», затем «ОК» и операционная система создаст новый файл с именем сценария и расширением wsh. Двойной щелчок по этому файлу запустит исполнение сценария с параметрами, установленными на вкладке свойств сценария.

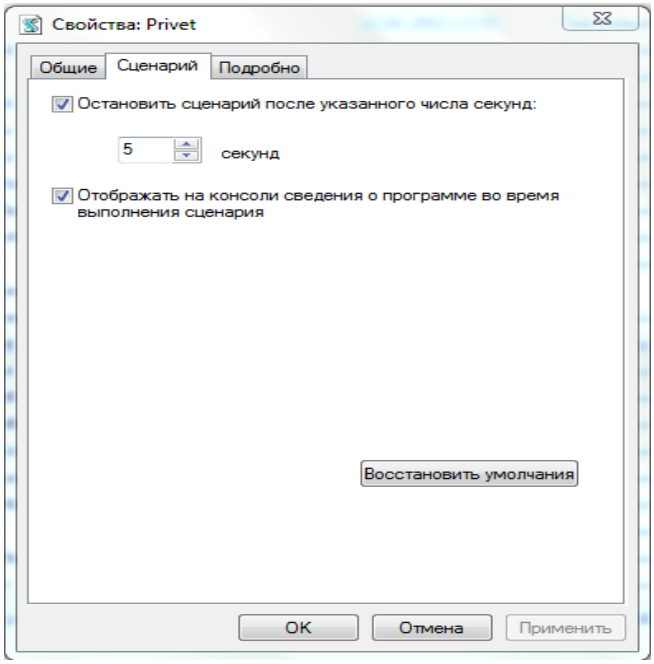


Рис. 1.6. Параметры сервера заданы
из меню пункт Properties (Свойства)

Файлы с расширением wsh представляют собой текстовые файлы, похожие на ini – файлы ОС. Содержание такого файла можно получить, открыв файл с помощью текстового редактора Блокнот (рис. 1.7).

```
[ScriptFile]
Path = F:\PRIMER_WSH\Privet.vbs
[Options]
TimeOut = 5
DisplayLogo = 1
```

Рис. 1.7. Содержание файла с параметрами сервера
для сценария Privet.vbs

Оператор Path в разделе [ScriptFile] идентифицирует файл, в котором находится подлежащий исполнению сценарий. Свойства времени исполнения сценария определяются ключевыми словами в разделе [Options]. Величина Timeout равна значению, заданному пользователем на вкладке свойства сценария. Параметр DisplayLogo=1 активизирует вывод заставки в окне командной строки.

Вопросы для самопроверки

1. Как можно вывести на экран все параметры консольного сервера WSH?
2. Какое расширение имеет файл ярлыка сценария WSH?
3. Приведите структуру файла-ярлыка, содержащего параметры сервера WSH.
4. Какой параметр позволяет управлять временем вывода на экран диалогового окна сценария WSH?

1.3. Передача аргументов файлу сценариев

Для работы сценария из предыдущего примера никаких аргументов не требовалось. Однако чаще всего пользователь должен передавать сценарию один или несколько аргументов, таких как имя файла, значения переменных и др. На рис. 1.8 приведен текст программы Primer1.vbs, которая выводит все аргументы сценария на экран.

```
'*********************************************************************************************
' Язык - VBScript
' Имя файла – Primer 1.vbs
' Автор Иванов И.И.
' Дата – 12.02.2013
' Пример 2 - Сценарий предусматривающий передачу параметров
'*********************************************************************************************
text = "Передача аргументов"&VbCrLf    ' VbCrLf – перевод строки
set objArgs = WScript.Arguments         ' создаем объект
for i = 0 to objArgs.Count – 1          ' цикл обработки всех аргументов
text = text & objArgs(i) & VbCrLf       ' запись i –го аргумент в text
next                                    ' переход к (i + 1)-му аргументу
WScript.Echo text                       ' вывод аргументов на экран
```

Рис. 1.8. Листинг сценария вывода аргументов на экран

При запуске этого сценария из меню «Пуск» в окне «Выполнить» кроме файла сценария необходимо указать аргументы к этому сценарию. Разделителем аргументов служит символ пробела. Пример запуска сценария и результаты его работы приведены на рис. 1.9. Следует отметить, что слова «Привет всем» заключены в кавычки, чтобы передать строку с пробелами как один аргумент, иначе они воспринимаются WSH как отдельные аргументы.

Передача аргументов при запуске сценариев из командной строки осуществляется аналогично, учитывая замечания об использовании двойных кавычек.

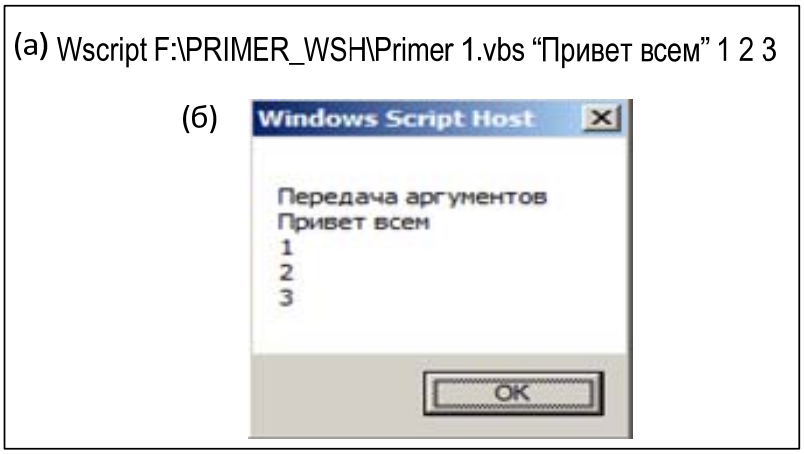


Рис. 1.9. Запуск сценария с аргументами – а
и результаты работы сценария – б

Передавать аргументы через диалоговое окно «Пуск» не совсем удобно. Иногда, особенно в случаях, когда значения аргументов изменяются редко, удобнее использовать для передачи аргументов ярлык файла сценариев. Для задания значения аргументов как свойства ярлыка необходимо:

- щелкнуть правой кнопкой значок файла сценариев и выбрать из контекстного меню команду Create Shortcut (Создать ярлык);
- далее щелкнуть правой кнопкой мыши по значку появившегося файла ярлыка и выбрать из контекстного меню Properties (Свойства);
- в текстовом поле Target (Объект) страницы свойств ярлыка необходимо добавить к команде запуска скрипта нужные аргументы.

ОС сохранит аргументы после того как окно свойств будет закрыто. После этого, если щелкнуть два раза по ярлыку сценария, то он запустится, и указанные в свойствах аргументы буду переданы сценарию.

Вопросы для самопроверки

1. Какой элемент служит разделителем при задании аргументов для сценариев WSH?
2. Каким образом передать строку с пробелами как один аргумент сценария WSH?
3. Как задать аргументы сценария при его запуске из меню ПУСК?
4. Как задать аргументы сценария при его запуске из командной строки?

1.4. Рекомендуемая структура сценария

Требования обеспечения «читаемости» сценариев, необходимость организации процедур их отладки с использованием средств отладки обусловливает необходимость выполнения при написании сценариев определенных требований к структуре сценариев.

В общем случае каждый сценарий должен содержать:

- информационный блок;
- блок инициализации;

- главное тело сценария;
- функции и процедуры.

В информационном блоке должна содержаться следующая информация: используемый для написания сценария язык (VBS), имя файла сценария, имя автора, дата создания, наименование сценария с описанием выполняемых функций.

Блок инициализации содержит объявления компонентов уровня сценария: переменные, константы, имена объектов (желательно с комментариями).

В главном теле сценария находится текст сценария, из него осуществляются вызовы функций и процедур.

В блоке функции и процедуры описываются используемые в главном теле сценария функции и процедуры.

```
' ========================= Script Information Header =====================
' Script Name:       ' название сценария;
' Date:              ' дата изменения;
' Author:            ' автор;
' Description:       ' описание;
' ========================= Initialization Block ============================
Option Explicit      ' директива, запрещающая автоматическое создание переменных
Dim <Variable>       ' объявление переменных;
Dim <Array>()        ' объявление массивов;
Set <Object>         ' подключение объектов;
' ========================= Script Main Logic ===============================
                     ' тело сценария;
' ========================= Functions and Procedures ========================
                      ' процедуры и функции.
```

Рис. 1.10. Рекомендуемая структура листинга сценария

Для того чтобы было удобно следовать приведенным рекомендациям, при создании сценариев желательно использовать шаблоны. В качестве шаблона возможно использование простых текстовых файлов (рис. 1.10).

В угловых скобках указаны обязательные команды и параметры, значения которых изменяются в зависимости от контекста сценария.

Вопросы для самопроверки

1. Какова общая структура файлов сценариев WSH?
2. Каково содержимое информационного блока файла сценария WSH?
3. Каково содержимое блока инициации файла сценария WSH?
4. В каком блоке находится текст сценария WSH?

2. ОСНОВЫ ПРОГРАММИРОВАНИЯ СЦЕНАРИЕВ

2.1. Классы и объекты в VBScript

VBScript является объектно-ориентированным языком, т. е. основной концепцией программирования на этом языке является понятие объектов и классов.

Классы – это блоки функциональности, которые используются в программах, в нашем случае – в программах сценариев. При программировании классы являются «шаблонами» для создания объектов.

На основе этих «шаблонов» в программах создаются отдельные экземпляры классов – объекты. Коллекция используемых классов обычно называется библиотекой типов и в Windows она хранится в файлах с расширением *.dll или *. осх. Такие библиотеки типов откомпилированы, классы из них можно использовать, но просмотреть их исходный код нельзя (используется технология «черного ящика»).

Обычно в программе сценария создается объект определенного класса (instantiation) и далее работа производится с этим объектом. В одном сценарии вполне можно использование несколько различных объектов одного и того же класса.

Создание объекта производится следующим образом:

```
Set Объектная_переменная = CreateObject("Библиотека.Класс").
```

Например:

```
Set ONetwork = CreateObject("Wscript.Network"),
```

где CreateObject() – функция, встроенная в VBScript.

Wscript – это одновременно и имя библиотеки типов (как в примере), и имя специального объекта, который создается в момент запуска скрипта (поэтому его специально создавать не нужно), и существует до конца выполнения скрипта WSH. Так в любом месте сценария можно использовать метод этого объекта Wscript.Echo.

Исполняя метод CreateObject, интерпретатор через ProgID получает из реестра путь к файлам библиотеки. Затем в память процесса загружается экземпляр объекта (как часть библиотеки) и его исходящий интерфейс подключается к сценарию. Эти действия прозрачны (невидимы) для пользователя. Пользователю необходимо знать только то, что метод CreateObject создает объект и присваивает ссылку на данный объект переменной-объекту, заданной в записи оператора. Эта переменная-объект определяет возможность доступа к методам и свойствам объекта, кроме того, ее можно использовать при последующих вызовах метода CreateObject, чтобы работать с вложенными объектами.

Чтобы явно удалить объект (например, для того, чтобы освободить оперативную память) можно использовать команду:

```
Set объект = Nothing.
```

Все объекты, которые не удалены явным образом, удаляются автоматически при завершении работы скрипта, однако правило хорошего тона предполагает удалять все объекты, в которых пропала надобность, явно (во избежание различных проблем).

Обычно в скрипте используются методы и свойства созданного объекта.

Метод может выполнять какие-либо действия, принимать и возвращать значения. Чаще всего неизвестно, какой именно код выполняет тот или иной метод используемого объекта, гораздо важнее знать, как тот или иной метод можно вызвать и использовать.

Существует три способа вызова метода объекта.

1. Простой вызов метода выглядит следующим образом:

```
Объект.метод.
```

При этом не возвращаются и не принимаются никакие параметры.

2. Вызов метода с указанием параметров:

```
Объект.метод параметр1 [, параметр2, ... , параметр].
```

Параметры указываются перечислением через запятую (скобки не используются).

3. Вызов метода с указанием параметров и запоминанием возвращаемого значения:

```
Моя_переменная = объект.метод (параметр1 [, параметр2, ... , параметрN])
```

В этом случае значение, которое возвращает метод, присваивается переменной. При этом применение круглых скобок для передаваемых параметров обязательно.

Даже если никакие параметры не передаются, круглые скобки все равно обязательны:

```
Моя_переменная = объект.метод().
```

Обращение к данным, которыми обладает объект, обычно производится через свойства этого объекта.

Через свойства можно получать информацию о данных объекта, а также изменять эти данные. Самый простой синтаксис для изменения свойств объекта выглядит следующим образом:

```
Объект.Свойство = Значение.
```

Значение может быть обычной константой (10 или «Сообщение»), простым выражением (10 + 5), свойством другого объекта:

Объект1.Свойство = Объект2.Свойство,

возвращаемым значением какого-либо метода:

Объект.Свойство = Объект2.Метод().

Свойство объекта можно запомнить в переменной:

Переменная = Объект.Свойство.

Требуемый уровень функциональности сценариев обеспечивается наличием обширной библиотеки классов. В Windows, как правило, библиотеки классов реализованы с помощью COM-технологий. Все классы идентифицируются ProgID (Programmatic Identifier). Информация о ProgID хранится в разделе реестра HKEY_CLASSES_ROOT и через него определяется, какой файл *.dll, *.exe и *.ocx содержит данный класс. ProgId состоит из имени библиотеки типов и имени класса. Например, при выполнении команды:

Set MyObject = CreateObject(“Scripting.FileSystemObject”)

через реестр определяется физический файл, в котором находится библиотека Scripting и класс FileSystemObject.

Создание сценариев в среде WSH требует учета ограничения, которые существуют при работе с COM-объектами.

Одно из главных ограничений – это ограничение по типам данных, которые могут передаваться COM-объектам, т. е. из скриптов можно обращаться далеко не ко всем COM-объектам. COM-объекты, которые полностью соответствуют требованиям среды WSH, называются объектами автоматизации (*automation objects*). Ниже перечислены наиболее важные библиотеки/объектные модели, обычно используемые в целях администрирования ОС:

Windows Script Host Object Model (wshom.exe) – работа с сетью, ярлыками, средой Windows и т. п.;

Microsoft Scripting Runtime (scrrun.dll) – классы для работы с файлами, папками, дисками, шифрованием и т. п.;

Microsoft ADO (набор файлов, начинающийся на msado) – классы для работы с базами данных;

Microsoft SQLDMO Object Library (файл sqldmo.dll) – набор классов для получения полного контроля над Microsoft SQL Server (возможность производить любые административные операции, выполнять запросы и т. п.);

Microsoft CDO – (файлы olemsg.dll, cdonts.dll, cdosys.dll) – наборы классов для работы с электронной почтой;

Microsoft Office 11.0 Object Library (mso.dll) и сопутствующие ей библиотеки отдельных приложений Office – наборы классов для работы с приложениямиOffice;

Internet Explorer Object Library (iexplore.exe) – библиотека классов для работы с Internet Explorer;

Active Directory Scripting Interface (adsldp.dll, wldap32.dll, adsnt.dll, adsnds.dll, adsnw.dll) – взаимодействие с объектами в каталогах Active Directory, NT, NetWare и т. п.;

Microsoft WMI Scripting v1.1 (wbemdisp.tlb) – расширение возможностей программ через программный интерфейс WMI.

Рассмотрим пример использования объекта Wscript для определения и вывода на экран его свойств с использованием одного из методов. Создим скрипт WscriptProperties.vbs, в котором методом MsgBox выводятся все свойства специального объекта WScript, за исключением Arguments, StdIn, StdErr, StdOut.

```
'*********************************************************************************************************
'Язык -VBS
'Имя файла - WscriptProperties.vbs
'Автор – учебный
'Дата – 20.11.2010
'Пример 3 – вывод на экран методом MsgBox всех свойств
 'специального объекта WScript, за исключением Arguments, StdIn, StdErr, 'StdOut.
'*********************************************************************************************************
MsgBox ("Application: " & WScript.Application & vbCRLF & _
"FullName: " & WScript.FullName & vbCrLf & _
"Name: " & WScript.Name & vbCrLf & _
"Path: " & WScript.Path & vbCrLf & _
"ScriptFullName: " & WScript.ScriptFullName & vbCrLf & _
"ScriptName: " &  WScript.ScriptName & vbCrLf & _
"Version: " & WScript.Version & vbCrLf)
```

Рис. 2.1. Листинг сценария определения свойств объекта WScript

Один из вариантов реализации требуемого сценария показан на рис. 2.1, а результаты выполнения сценария WscriptPropertie на рис. 2.2.

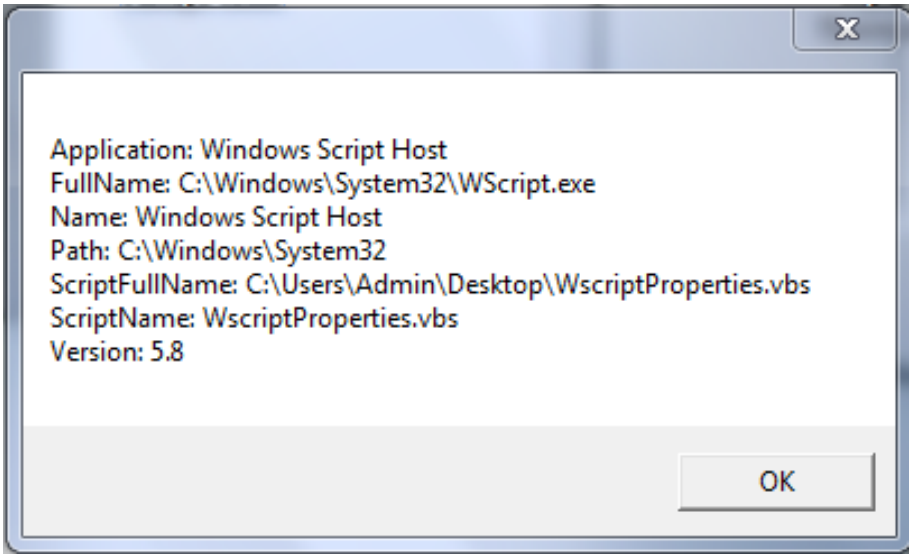


Рис. 2.2. Результаты выполнения сценария WscriptProperties.vbs

Очень важно уметь находить информацию о классах и тех свойствах и методах этих классов, которые можно использовать в скриптах.

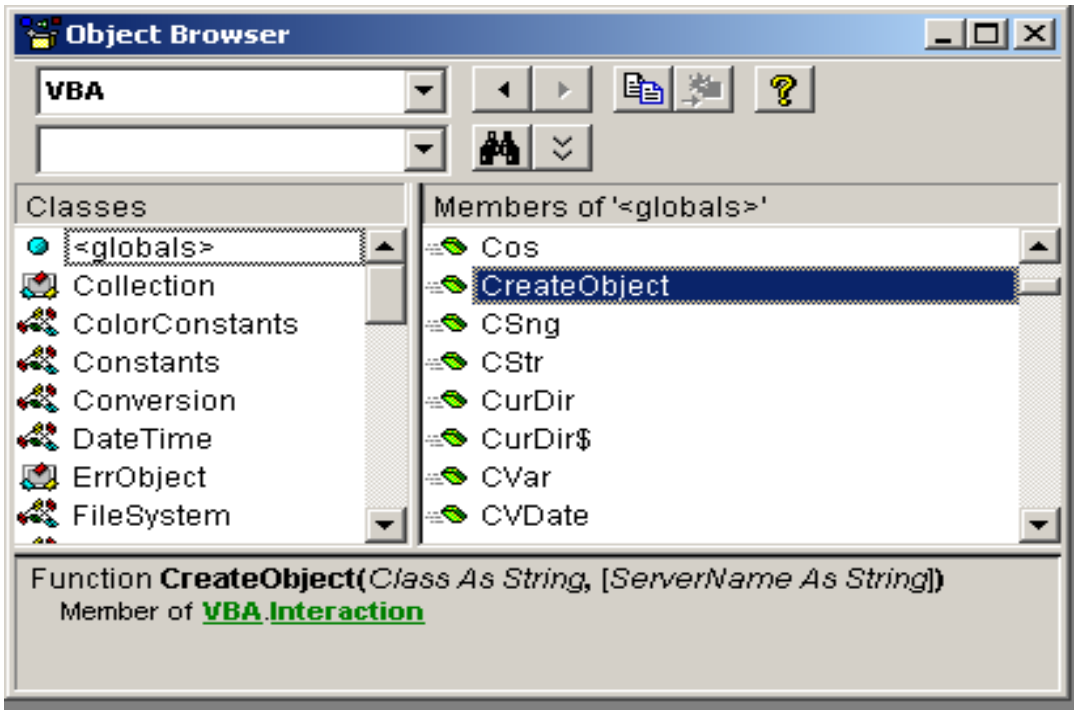


Рис. 2.3. Основные классы VBA

Для этой цели можно использовать встроенный в VBA просмотрщик, который поставляется вместе с MS Office. Так, в Word 2000 необходимо нажать Alt-F11, а затем F2. Результат такой операции представлен на рис. 2.3.

Вопросы для самопроверки

1. Дайте определение объектов, используемых в сценариях WSH.
2. Как создать объект в сценарии WSH?
3. Какой объект создается автоматически, в момент запуска сценария WSH?
4. Дайте определение метода объекта сценария WSH.
5. Какие способы вызова метода объекта используются в сценариях WSH?
6. Что такое свойство объекта?
7. Как можно изменить свойство объекта?

2.2. Формирование и вывод сообщений на экран

Сформированные в сценарии строки текста можно выводить в стандартный выходной поток (в консольном режиме) или в диалоговое окно (с использованием графического интерфейса) следующими способами:

– с помощью метода Echo объекта Wscript;
– с помощью методов Write и WriteLine объекта Wscript.StdOut;
– с помощью встроенной в VBS функции MsgBox;
– с помощью метода Popup объекта WshShell.

Метод Wscript.Echo

Рассмотрим использование метода Wscript.Echo (рис. 2.4) для вывода информации на экран. Для корректного отображения символов кириллицы они должны быть представлены в Windows-кодировке (CP 1251).

```
'*****************************************************************
' Язык -VBS
' Имя файла - Echo.vbs
' Автор – учебный
' Дата – 20.11.2010
' Вывод эхо – вывод на экран текста методом Echo
'*****************************************************************
 Wscript.Echo "Использование метода Wscript.Echo"
 Wscript.Echo " 2 + 3 = ", 2+3
```

Рис. 2.4. Листинг сценария
с использованием метода WScript.Echo

Запуск сценария из проводника приводит к поочередному выводу на экран каждой строки в окне с одной кнопкой ОК (для вывода следующей строки необходимо нажать эту кнопку). Результат работы сценария представлен на рис. 2.5.

Рис. 2.5. Результаты работы сценария Echo.vbs

Запуск сценария из командной строки приводит к выводу указанных строк в окно командной строки.

Задача вывода одним оператором Wscript.Echo нескольких строк текста решается предварительным формированием этих строк и записи их в одну переменную с последующим выводом на экран этой переменной. При этом используется предопределенная переменная vbCrLf – вывод, которой соответствует переводу строки. Сценарий, решающий указанную задачу, приведен в примере рис. 2.6.

```
'*********************************************************************
' Язык -VBS
' Имя файла – Echo1.vbs
' Автор – учебный
' Дата – 20.02.2013
' Вывод эхо – вывод на экран нескольких строк
'*********************************************************************
Option Explicit              ' требует явного объявления переменных
Dim s                        ' объявляем переменную S
s = "Вот" & vbCrLf & "несколько" & vbCrLf & "строк"
Wscript.Echo s               ' вывод на экран строк
```

Рис. 2.6. Листинг сценария для вывода нескольких строк

Результат работы Echo1.vbs сценария представлен на рис. 2.7.

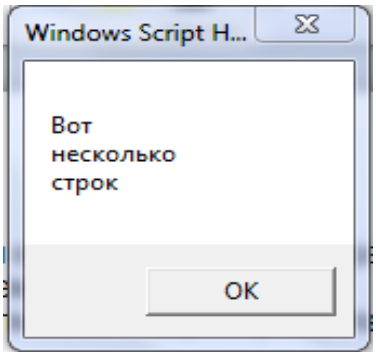


Рис. 2.7. Результат работы сценария Echo1.vbs

Стандартный выходной поток StdOut

Вывод строк в сценариях, исполняемых в консольном режиме, можно осуществить с использованием стандартного выходного потока Wscript.StdOut. Запускать сценарии, в которых используется стандартный выходной поток, следует только из командной строки. Запуск такого сценария с помощью Wscript.exe приведет к ошибке.

В примере рис. 2.8 показан способ использования выходного потока при выводе информации на экран.

```
'*************************************************************************
' Язык - VBS
' Имя файла - StdO.vbs
' Автор - учебный
' Дата -20.11.2010
'Пример - вывод на экран строк с использованием StdOut
'*************************************************************************
Option Explicit          'требует явного объявления переменных
Dim s                    ' объявляем переменную
s=5+2
Wscript.StdOut.Write"Использование метода StdOut"
Wscript.StdOut.WriteLine "StdOut.WriteLine "      'печать строки
Wscript.StdOut.WriteLine "Пример, 5+2="&s
```

Рис. 2.8. Листинг сценария с использованием стандартного выходного потока

Для создания более компактного текста сценария можно сохранить ссылку на стандартный выходной поток Wscript.StdOut в отдельную переменную, а затем при вызове методов Write и WriteLine использовать эту переменную (рис. 2.9). Результат работы сценария представлен на рис. 2.10.

```
'*************************************************************************************
'Язык - VBS
'Имя файла - StdO1.vbs
'Автор - учебный
'Дата -20.11.2010
'Пример -  использование ссылки на  Wscript.StdOut
'*************************************************************************************
Option Explicit   'требует явного объявления всех переменных
Dim s,StdO ' объявляем переменные
s=5+2
Set StdO=WScript.StdOut 'сохраняем ссылку на StdOut в переменной
StdO.WriteLine"Пример использование метода WriteLine"
StdO.WriteLine "Пример, 5+2="&s
```

Рис. 2.9. Листинг сценария с использованием ссылки на выходной поток

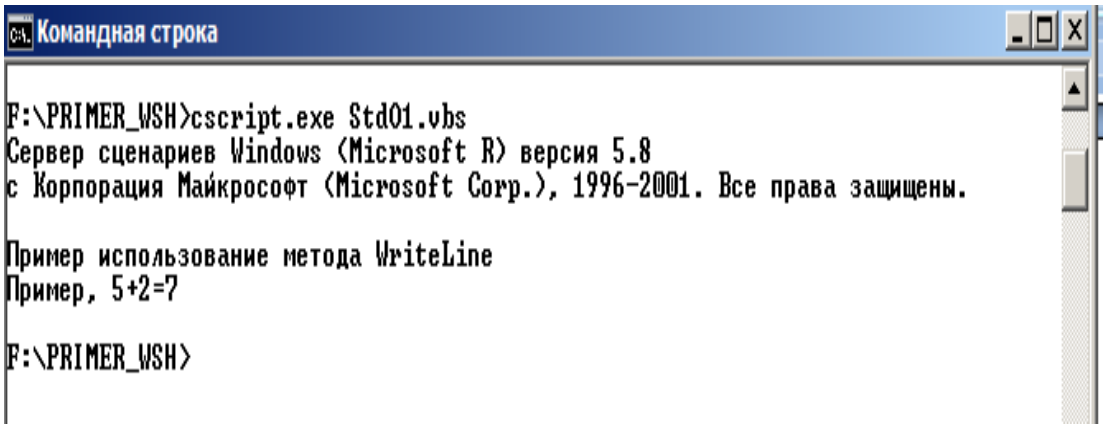


Рис. 2.10. Результаты работы сценария StdO1.vbs

Функция VBScript MsgBox

В языке VBScript существует специальная функция MsgBox, которая позволяет выводить информацию в диалоговое окно с несколькими кнопками. В этом окне можно также задать заголовок окна и вид кнопки.

Обращение к функции имеет следующий вид:

MsgBox(promt [,buttons][,title][,helpfile, context]),

где prompt – задает сообщение в диалоговом окне, buttons – определяет, какие именно кнопки выводятся в окне, title – задает заголовок (наименование) окна. Параметр helpfile задает путь к файлу помощи, а context определяет идентификатор содержания помощи.

Параметр buttons может быть задан цифрой или соответствующей переменной:

0 или vbOkOnly – выводится кнопка ОК;

1 или vbOkCancel – выводятся кнопки ОК и Cancel;

2 или vbAbortRetryIgnore – выводятся кнопки Abbort (Стоп), Retry (Повторить) и Ignore (Пропустить);

3 или vbYesNoCancel – выводятся кнопки Yes (Да), No (Нет) и Cancel (Отменить);

4 или vbYesNo – выводятся кнопки Yes и No;

5 или vbRetryCancel – выводятся кнопки Retry и Cancel.

Пример использования функции MsgBox для организации диалога приведен (рис. 2.11).

```
'*********************************************************************************************
' Язык - VBS
' Имя файла - MBox.vbs
' Автор - учебный
' Дата -20.11.2010
' Пример -  задание диалогового окна с кнопками
'*********************************************************************************************
Option Explicit                 ' требует явного объявления всех переменных
Dim Res,Txt,Tit                 ' объявляем переменные
Txt="Нажмите любую из кнопок"
Tit="Пример вызова окна с кнопками с использованием MsgBox"
Res=MsgBox(Txt,1,Tit)      ' вызов функции диалогового окна
If Res=vbOk Then
   Wscript.Echo "Нажата кнопка OK, ","Возвращаемое значение =",Res
Else
   Wscript.Echo "Нажата кнопка Cancel, ","Возвращаемое значение =",Res
End If
```

Рис. 2.11. Листинг сценария с использованием функции MsgBox

Результаты работы сценария Mbox.vbs представлены на рис. 2.12.

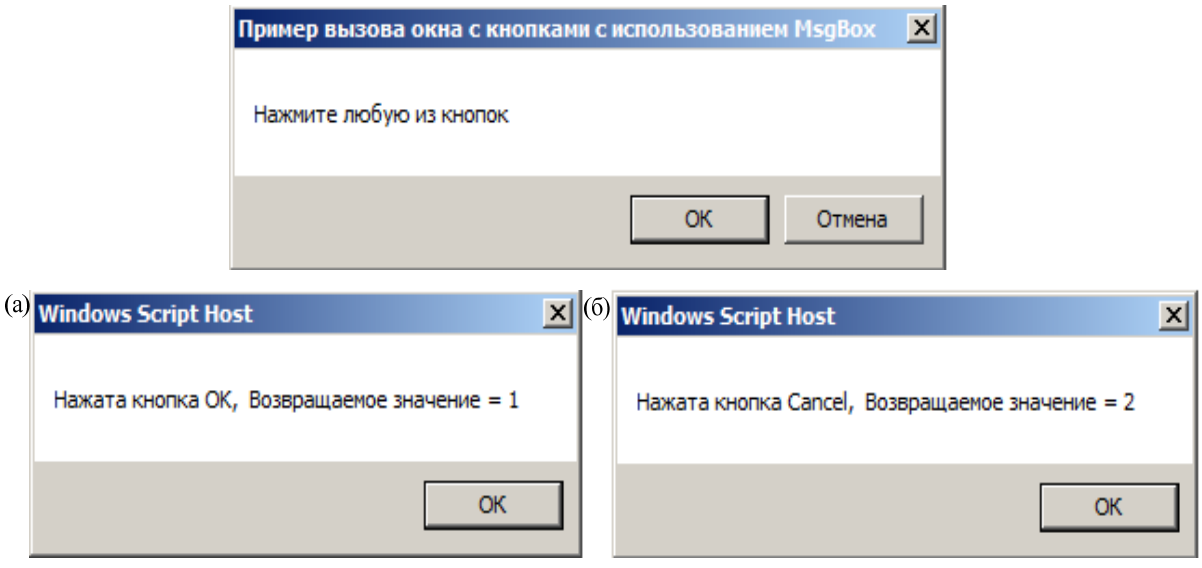


Рис. 2.12. Результаты работы сценария Mbox.vbs:
а – нажата кнопка OK; б – нажата кнопка Отмена

Значением функции MsgBox является константа, соответствующая нажатой в диалоговом окне кнопки (в примере такими константами являются vbOk и vbCancel). Таким образом, MsgBox используется в сценариях

для выбора пользователем одного из возможных вариантов. Недостатком функции является ограничение наименований кнопок, их нельзя задавать произвольным образом. Допустимыми наименованиями кнопок являются ОК, Отмена, Стоп, Повтор, Пропустить, Да и Нет.

Метод Popup объекта WshShell

С помощью метода Popup можно создавать такие же диалоговые окна, как и с помощью функции MsgBox. Popup входит в состав объектной модели WSH и поддерживается объектом WshShell. Применять метод для создания диалоговых окон позволяет следующий оператор:

```
Объект.Popup параметры
```

Объект Wscript не поддерживает метод Popup, а WSH не предоставляет объект WshShell автоматически. Следовательно, прежде чем использовать метод Popup, необходимо создать экземпляр объекта WshShell и получить на него ссылку. Как уже отмечалось, в сценариях WSH для создания объектов служит метод CreateObject объекта Wscript. Параметром CreateObject является ProgID запрашиваемого объекта. ProgID объекта WshShell определен как WScript.Shell. Метод CreateObject ищет нужный объект в реестре. Затем экземпляр этого объекта загружается в оперативную память, и CreateObject возвращает ссылку на этот объект. Ссылка присваивается переменной – в нашем случае переменной WShell:

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

После создания переменной-объекта можно для вывода диалоговых окон использовать метод WShell.Popup.

Синтаксис вызова метода имеет следующий вид:

```
Res=WShell.Popup(текст, таймаут, заголовок, кнопки),
```

где текст – содержит текст, который выводится в диалоговом окне. Этот текст можно сформировать конкатенацией констант, подстрок и переменных. Таймаут – определяет значение таймаута диалогового окна. Если пользователь до истечения таймаута не закроет окно щелчком кнопки, сценарий закроет окно автоматически. Если задать значение 0, таймаут будет отключен и диалоговое окно будет открыто, пока его не закроет пользователь. Заголовок – задает текст заголовка окна. Параметр кнопки – определяет вид значка и набор кнопок диалогового окна. Значения этого параметра такие же, как и у одноименного параметра функции MsgBox.

После закрытия диалогового окна метод Popup возвращает код кнопки, которой пользователь закрыл окно. Возвращаемые рассматриваемым методом значения представлены в табл. 2.1.

Таблица 2.1

Перечень возвращаемых методом Popup значений

Значение	Константа VBS	Описание
–1	–	Кнопка не нажата за время nSecToWait
1	vbOk	Нажата кнопка Ok
2	vbCancel	Нажата кнопка Отмена (Cancel)
3	vbAbort	Нажата кнопка Стоп (Abort)
4	vbRetry	Нажата кнопка Повтор (Retry)
5	vbIgnore	Нажата кнопка Пропустить (Ignore)
6	vbYes	Нажата кнопка Да (Yes)
7	vbNo	Нажата кнопка Нет (No)

Пример использования метода Popup для организации диалога приведен в сценарии Pop.vbs (рис. 2.13).

```
'*********************************************************************************************
'Язык - VBS
'Имя файла - Pop.vbs
'Автор - учебный
'Дата -20.11.2010
'Пример -  задание диалогового окна с кнопками методом Popup
'*********************************************************************************************
Option Explicit   'требует явного объявления всех переменных
Dim Res,WshShell ' объявляем переменные
Set WshShell=WScript.CreateObject("WScript.Shell") ' создали объект
Res=WshShell.Popup("Нажмите кнопку",0, "Пример использования Popup", _
 vbOkCancel + vbInformation) ' вызов функции диалогового окна
Wscript.Echo"Возвращаемое значение  равно ", Res 'печать значения функции
```

Рис. 2.13. Листинг сценария Pop.vbs

Результаты работы сценария Pop.vbs аналогичны результатам работы сценария Mbox.vbs, представленным на рис. 2.12.

Вопросы для самопроверки

1. Назовите основные методы, используемые в сценариях WSH для вывода информации на экран монитора.
2. Каким образом, используя метод Wscript.Echo, вывести на экран сообщение, содержащее несколько строк?
3. Как должен запускаться сценарий WSH, содержащий метод Wscript.StdOut (стандартный выходной поток)?
4. Каково назначение функции MsgBox() в сценариях WSH?
5. Какой объект WSH поддерживает метод Popup?
6. Для чего используется метод Popup в сценариях WSH?

2.3. Формирование и ввод сообщений в сценарий

Для организации полноценного диалога пользователя со сценарием необходимо иметь возможность передавать внутрь сценария вводимую с клавиатуры информацию. Ввод информации можно осуществить при запуске сценария с помощью cscript.exe, в этом случае для ввода информации используется стандартный входной поток StdIn. При использовании графического интерфейса (wscript.exe) необходимо применять функцию языка VBS – InputBox.

Ввод строк в консольном режиме

Ввод строки в консольном режиме обеспечивает метод WScript.StdIn.ReadLine, при этом ввод завершается нажатием клавиши <Enter>.

Необходимо отметить, что при использовании стандартного входного потока WScript.StdIn в Windows XP возникает проблема кодировки символов кириллицы. Метод WScript.StdIn.ReadLine возвращает строку в DOS-кодировке, а методы вывода строки на экран (WScript.StdOut.WriteLine или WScript.Echo) используют Windows-кодировку. Для корректного отображения введенных символов кириллицы на экране необходимо использовать дополнительные функции конвертации из DOS в Windows-кодировку. Стандартных методов или функций для реализации конвертации в VBS не существует, поэтому эту задачу следует запрограммировать отдельно.

Рассмотрим сценарий, позволяющий осуществить ввод одной строки символов с последующим выводом этой строки на экран. Вначале запрограммируем функцию конвертации кодов каждого символа строки – Function DosToWin(s). В качестве аргумента функции используется строка символов – s, сформированная с помощью метода WScript.StdIn.ReadLine:

```
'Имя файла - Stdin.vbs
'Ввод строки символов, перекодировка, вывод на экран
Function DosToWin(s)
Dim i, k, ss=""
For i=1 To Len(s) 'цикл по всем символам строки s
k=Asc(Mid(s,i,1)) 'определяем ANSI код i-го символа
'далее изменяем код k на код соответствующего символа
'в Windows кодировке
If (128<=k) And (k<=175) Then
k=k+64
ElseIf (224 <=k) And (k<=239) Then
k=k+16
ElseIf k=240 Then
k=168
ElseIf k=241 Then
```

```
k=184
End If
ss=ss+Chr(k)
Next
DosToWin=ss 'возвращаем преобразованную строку
End Function
'_____Начало  сценария_____
Dim s
Wscript.StdOut.Write"Введите одну строку: "
s=WScript.StdIn.ReadLine 'Вввод строки с клавиатуры
WScript.StdOut.WriteBlankLines 1 'вывод пустой строки
Wscript.StdOut.Write"Вывод строки без конвертации: "
WScript.StdOut.WriteLine s
WScript.StdOut.WriteBlankLines 1 'вывод пустой строки
WScript.StdOut.Write "Вывод строки после конвертации кодов:"
'Конвертируем коды введенной строки и выводим строку на экран
WScript.StdOut.WriteLine DosToWin(s)
```

Результаты работы сценария приведены на рис. 2.14. Еще раз отметим, запуск этого сценария следует производить из командной строки с использованием сервера Cscript.exe.

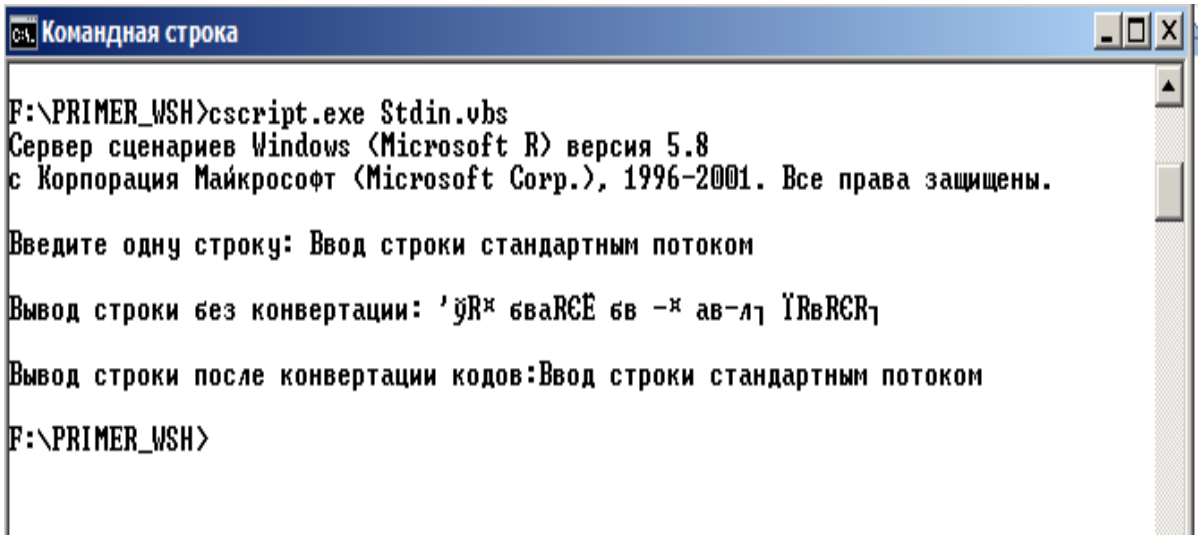


Рис. 2.14. Результаты работы сценария Stdin.vbs

При необходимости ввести несколько строк подряд используется метод Wscript.StdIn.ReadAll. Режим ввода информации в этом случае прерывается нажатием клавиш <Ctrl>+<Z>. Все вводимые строки упаковываются в одну переменную. Из этой переменной затем можно, использовав внутреннюю функцию VBS split, сформировать массив строк. Сценарий Stdinall.vbs, показывающий использование метода Wscript.StdIn.ReadAll, приведен ниже:

```
'Имя файла - Stdinall.vbs
'Ввод нескольких строк символов, без перекодировки,
'формирование массива строк и вывод их на экран
Dim s, ArrS, ColStr ' описание переменных
'печатаем приглашение для ввода
Wscript.StdOut.Write"Введите несколько строк: "
s=WScript.StdIn.ReadAll 'Вввод строк  с клавиатуры
WScript.StdOut.WriteBlankLines 3 'вывод 3 пустых строк
Wscript.StdOut.Write "s="&s' вывод значения переменной s
ArrS=Split(s, vbCrLf) ' формируем массив из введенных строк
ColStr=UBound(ArrS)+1'определяем количество строк в массиве
'Печатаем введенные строки
Wscript.StdOut.WriteLine "Всего введено строк " & ColStr
For i=1 To ColStr
WScript.StdOut.WriteLine i & ": " & ArrS(i-1)
Next
```

Результаты работы сценария Stdinall.vbs приведен на на рис. 2.15.

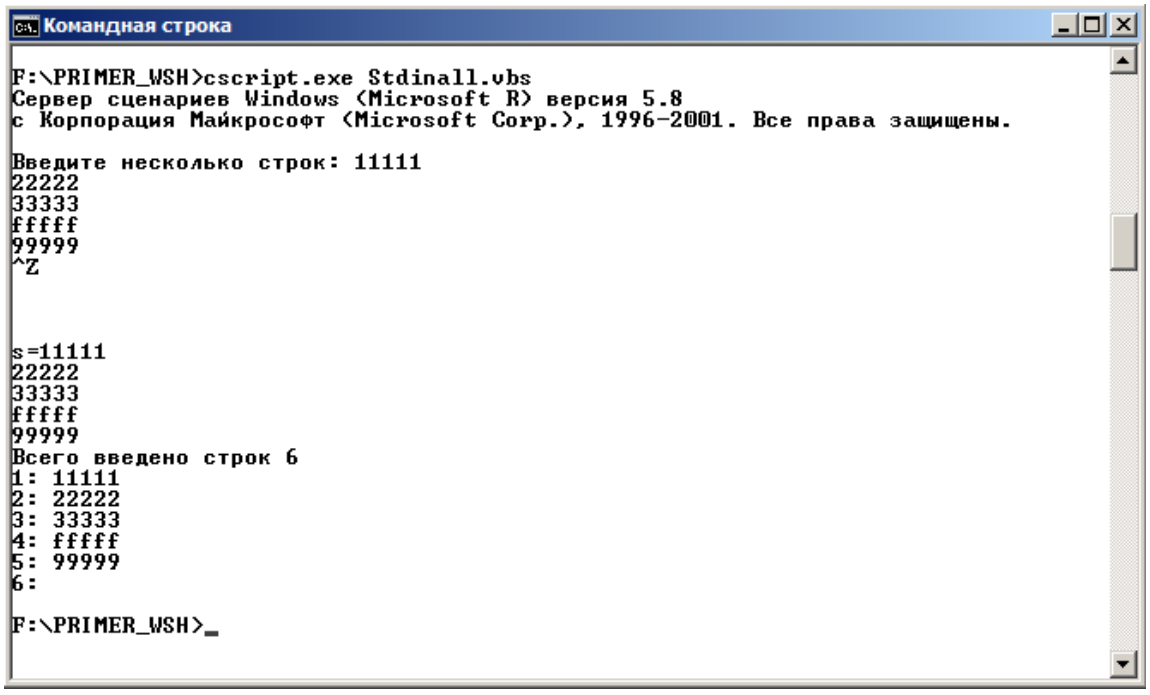


Рис. 2.15. Результаты работы сценария Stdinall.vbs

Ввод строк с использованием графического интерфейса

В сценариях VBScript информацию можно вводить с помощью диалогового окна, создаваемого функцией InputBox.

Синтаксис вызова функции имеет следующий вид:

InputBox(prompt [, title][, default][, xpos][, ypos][, helpfile][, context]),

где параметр prompt задает сообщение, которое печатается перед строкой ввода; title определяет заголовок диалогового окна; default – значение, которое выводится по умолчанию в строку ввода. Параметры xpos и ypos определяют координаты левого верхнего угла окна. При организации контекстно-зависимой помощи используются параметры helpfile – путь к файлу помощи и число context – идентификатор содержания помощи.

Пример использования функции IputBox для ввода информации приведен в сценарии Inbox.vbs (рис. 2.16), а результат его работы – на рис. 2.17.

```
'*********************************************************************************
'Язык - VBS
'Имя файла - Inbox.vbs
'Автор - учебный
'Дата -20.11.2010
'Пример -  использование функции InputBox для ввода данных
'*********************************************************************************
Dim s,s1 ' описание переменных
s1="Пример"& vbCrLf & " Диалогового окна"&vbCrLf &"Для ввода строки"
'Выведем диалоговое окно
s=InputBox(s1,"Диалоговое окно для ввода данных")
'Выведем на экран введенную ранее информацию
MsgBox "Было введено: "& s
```

Рис. 2.16. Листинг сценария Inbox.vbs использования функции InputBox

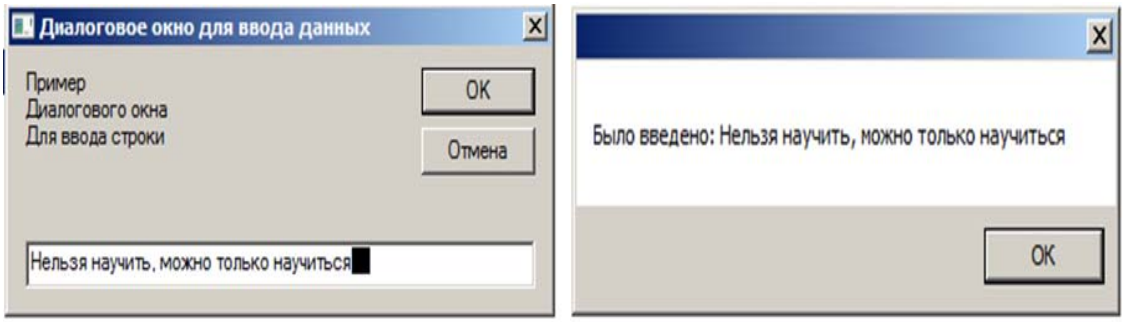


Рис. 2.17. Результаты работы сценария Inbox.vbs

Вопросы для самопроверки

1. Какой метод используется при вводе информации в консольном режиме исполнения сценария WSH?
2. В какой кодировке передает строку метод сценария WSH Wscript.StdIn.ReadLine при вводе информации?
3. Как должен быть закончен ввод информации, содержащей несколько строк, при использовании в сценарии WSH метода Wscript.StdIn.ReadAll?
4. Какая функция VBS может быть использована для ввода информации в режиме диалогового окна сценария WSH?

2.4. Доступ к файлам из сценариев WSH

Сценарии WSH позволяют получить полный доступ к файловой системе компьютера. Для работы с файловой системой используются восемь объектов, главным из которых является FileSystemObject. С помощью методов этого объекта можно выполнять следующие основные операции:

- копировать, перемещать и удалять файлы и каталоги;
- создавать и открывать текстовые файлы и каталоги;
- создавать объекты Drive, Folder, File.

Следует отметить, что у объекта FileSystemObject нет методов для доступа к двоичным файлам. Такие файлы можно читать и записывать, используя методы, предназначенные для работы с текстовыми файлами с последующим преобразованием информации.

Рассмотрим задачу считывания информации из текстового файла, имя которого зададим переменной file.

Чтобы прочитать данные из файла необходимо создать объект FileSystemObject. Для его создания внутри сценария воспользуемся методом CreateObject существующего объекта Wscript:

```
Set fso=Wscript.CreateObject("Scripting.FileSystemObject").
```

Переменная-объект fso позволяет в дальнейшем использовать все методы и свойства FileSystemObject. Далее следует проверить наличие файла, так как при его отсутствии исполнение сценария приведет к ошибке:

```
If fso.FileExists(file) Then ...
```

При наличии файла его следует открыть методом OpenTextFile объекта FileSystemObject:

```
Set txtStream=fso.OpenTextFile(file)
```

Этот метод требует передачи ему параметра file, содержащий путь и имя файла. Кроме этого параметра метод поддерживает еще несколько необязательных параметров:

```
Object_name.OpenTextFile(файл [, режим][, создать][, формат]),
```

где файл – имя существующего текстового файла.

Режим – параметр задает режим ввода/вывода информации в файл. Параметр может содержать константы ForReading (со значением 1, определяет доступ только для чтения), ForWriting (со значением 2, определяет

доступ только для записи), ForAppending (со значением 8, определяет доступ для записи и позволяет дописывать данные к концу файла).

Создать – необязательный параметр, содержит булево значение True (отсутствующий файл будет создан) или False (файл не создается).

Формат – параметр задает формат открываемого текстового файла. Если этот параметр не задан, для файла определяется формат ASCII. Значение TristateTrue (со значением –1) открывает файл в формате Unicode, TristateFalse (со значением 0) открывает файл в формате ASCII).

Чтобы прочитать строки из текстового файла, необходимо применить к объекту «файл» метод ReadLine. Чтобы при чтении не выйти за пределы файла, необходимо проверять, достигнут ли конец файла. Организация цикла, построчно читающего содержимое файла и помещающего его в переменную Text, выглядит следующим образом:

```
Do While Not (txtStream.atEndOfStream)
      Text=Text & txtStream.ReadLine & vbCrLf
 Loop
```

Проверку достижения конца файла осуществляет метод atEndOfStream. Если конец файла не достигнут, он возвращает False. После выхода из цикла в строковой переменной Text будет находиться все содержимое файла. После того как содержимое файла прочитано переменную Text можно обрабатывать дальше.

В рассматриваемом далее примере сценария по умолчанию будет прочитываться файл F:\PRIMER_WSH\Fread.txt. Если не указан иной файл, сценарий читает этот файл и выводит его содержимое. Предусмотрим в сценарии определение имени считываемого файла путем перетаскивания значка этого файла на значок файла сценария.

Все эти действия реализуются следующими операторами:

```
file="F:\PRIMER_WSH\Fread.txt" 'определение файла по умолчанию
Set objArgs=Wscript.Arguments   'создание объекта
If objArgs.Count > 0 Then           'аргумент найден
   file=objArgs(0)                  'установить имя файла
End If
```

После отработки указанного выше фрагмента сценария в переменной file будет имя F:\PRIMER_WSH\Fread.txt или путь и имя файла, значок которого перетащили на значок файла сценария.

При организации просмотра текстовых файлов с помощью диалоговых окон следует учитывать следующие особенности. Если выводить методом

Echo содержимое очень большого файла, то диалоговое окно будет больше рабочего стола. При этом конец текста и кнопка ОК будут скрыты за панелью задач. При использовании функции MsgBox текст большого файла будет выводиться не полностью, так как максимальная длина параметра prompt функции MsgBox равна 1024 символам. То есть при попытке вывести в окно MsgBox содержимое файла, в котором более 1024 символов, в окне появится лишь первая часть файла. Устранения указанных недостатков возможно использованием в качестве средства для просмотра содержимого текстового файла окна текстового редактора Notepad. Для этого следует создать уже известный объект WshShell, а для запуска редактора и загрузки в него читаемого файла необходимо воспользоваться методом этого объекта – Run, предварительно сформировав переменную s, которая используется в качестве параметра метода Run:

```
s="notepad " & file,
```

где file содержит путь и имя просматриваемого файла.

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

– создание объекта и

```
WshShell.Run s
```

– запуск блокнота и загрузка содержимого файла.

В следующем сценарии Rtxt.vbs пользователю предлагается просмотреть содержимое текстового файла, имя которого определяется при «натаскивании» значка текстового файла на значок файла сценария (Rtxt.vbs), при этом пользователь может определить в качестве средства для просмотра содержимого файла диалоговое окно или окно текстового редактора:

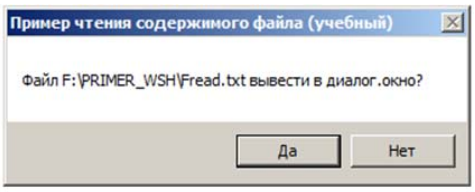
```
'Имя файла - Rtxt.vbs
'Чтение и вывод на экран текстового файла
Dim Text, Title,s
Dim fso, objArgs, WshShell ' Переменные-объекты
Dim txtStream 'Текстовы поток
Dim file ' имя файла

file="F:\PRIMER_WSH\Fread.txt" 'определение файла по умолчанию
Set objArgs=Wscript.Arguments 'создание объекта

If objArgs.Count > 0 Then 'аргумент найден
file=objArgs(0) 'установить имя файла равным значению первого аргумента
End If
Text="Содержимое файла " & file & vbCrLf & vbCrLf
```

```
Title="Пример чтения содержимого файла"
Set fso=Wscript.CreateObject("Scripting.FileSystemObject")'создать объект
'чтобы получить доступ к файловой системе
If fso.FileExists(file) Then    ' проверка - существует ли файл
Set txtStream=fso.OpenTextFile(file) ' открыть файл
Do While Not (txtStream.atEndOfStream)
Text=Text & txtStream.ReadLine & vbCrLf ' читать файл в переменную
Loop
Else
MsgBox "Файл : " & file & " не существует", vbOkOnly + vbCritical, Title
Wscript.Quit 1  ' завершить
End If
If MsgBox("Файл "& file &" вывести в диалог.окно?",vbYesNo, Title)=vbYes Then
MsgBox Text, vbOkOnly + vbInformation, Title 'вывести в окно
Else  ' вывести в блокнот
MsgBox "Файл " & file&"  выводится в блокнот", vbOkOnly + vbInformation, Title
Set WshShell = WScript.CreateObject("WScript.Shell") 'создаем объект
s="notepad " & file ' формируем s для запуска блокнота с загрузкой файла
WshShell.Run s ' запуск блокнота и загрузка  содержимого файла
End If
```

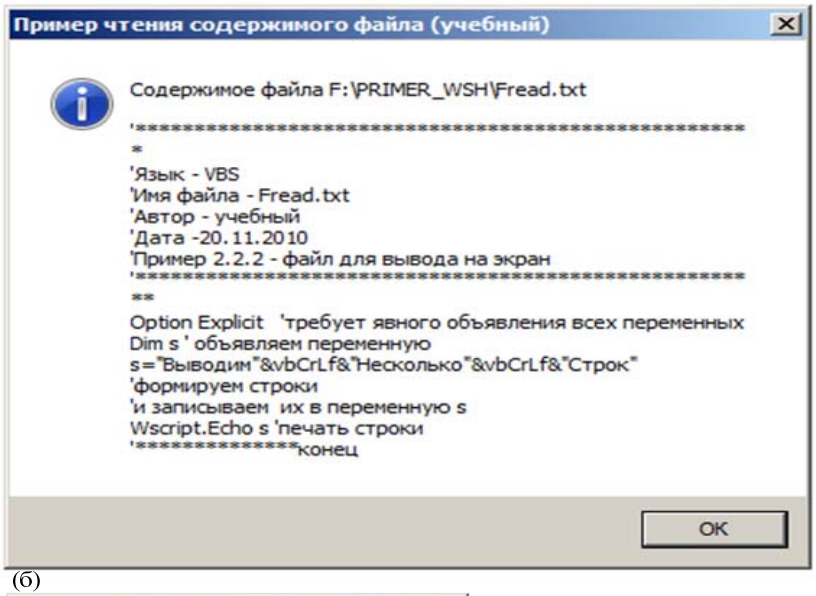
Результаты работы сценария Rtxt.vbs приведены на рис. 2.18. Рассмотрим задачу записи данных в текстовый файл. Данные, записываемые в файл, можно взять из другого файла или сформировать в сценарии. Рассмотрим работу сценария, который открывает файл-источник, копирует его в выходной файл и добавляет к его содержимому строку, сформированную в сценарии.

В качестве файла источника воспользуемся файлом f1.txt, который содержит 2 строки: «Это содержимое файла 1 – первая строка» и «Это содержимое файла 1 – вторая строка», в сценарии сформируем строку s= «А эта строка добавлена из сценария», в качестве выходного файла определим f2.txt.

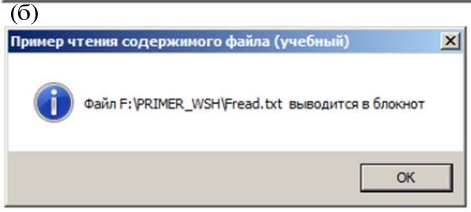
Чтобы получить доступ к текстовому файлу f1.txt необходимо создать объект FileSystemObject, далее файл должен быть открыт методом OpenTextFile. Файл f2.txt также необходимо открыть, причем он должен быть открыт в режиме, позволяющем производить в него запись информации, т. е. метод OpenTextFile должен быть использован с параметром ForWriting=2. Кроме того, если файл f2.txt отсутствует, то следует предусмотреть его создание, для этого при использовании метода OpenTextFile необходимо задать третий параметр True.



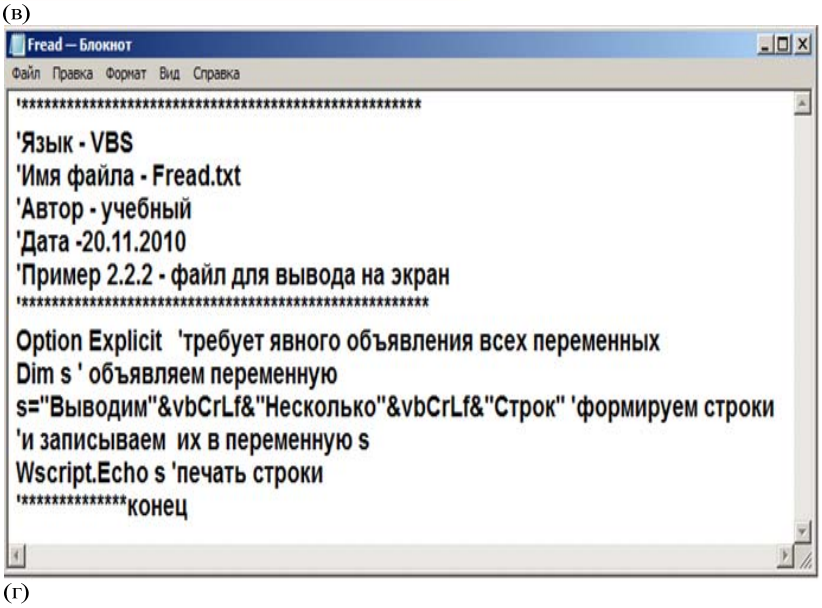
(а)



(б)



(в)



(г)

Рис. 2.18. Результат работы сценария Rtxt.vbs:

а – диалоговое окно, в котором следует определить метод вывода содержимого файла на экран; б – диалоговое окно вывода содержимого файла; в – диалоговое окно с предупреждением о запуске текстового редактора; г – окно текстового редактора с содержимым файла

Ниже приведен сценарий (Opwrfile.vbs), который решает поставленную задачу:

```
'Имя файла - Opwrfile.vbs
'Открыть файл, скопировать в него содержимое
'другого файла и добавить строку, сформированную в скрипте
Const file1="F:\PRIMER_WSH\f1.txt" 'исходный текстовый файл
Const file2="F:\PRIMER_WSH\f2.txt" 'выходной текстовый файл
Const ForWriting =2
Dim Text '
Dim fso ' Переменная-объект
Dim txtStream,  txtStreamOut  'Объекты TextStream
Set fso= CreateObject("Scripting.FileSystemObject") 'создали объект
If fso.FileExists (file1) Then    'проверили наличие файла - источника
  Set txtStream = fso.OpenTextFile (file1) 'открыли  исходный файл
  Set txtStreamOut = fso.OpenTextFile (file2, ForWriting, True) 'открыли
  'выходной файл
  Do While Not (txtStream.atEndOfStream)
     'построчно копируем f1.txt в f2.txt
     Text=txtStream.ReadLine  ' читать строку
      txtStreamOut.WriteLine Text ' записать строку
   Loop
   txtStreamOut.WriteLine "А эта строка добавлена из сценария"
  ' добавить строку к содержимому файла f2.txt
   Set txtStream=Nothing   ' освободить объекты
   Set txtStreamOut=Nothing
   Wscript.Echo "Файл " & file1 & " скопирован и дополнен в файл " & file2
Else
    Wscript.Echo "Файл " & file1 & " не найден "
End If
```

Результаты работы сценария Opwrfile.vbs приведены на рис. 2.19. При необходимости обеспечить добавление информации к уже существующему файлу (при ведении журналов, формировании Log-файлов и др.) можно использовать известный метод WriteLine, поддерживаемый объектом FileSystemObject. В этом случае открытие файла следует производить с параметром ForAppend=8, наличие третьего параметра True позволяет создать файл журнала при его отсутствии.

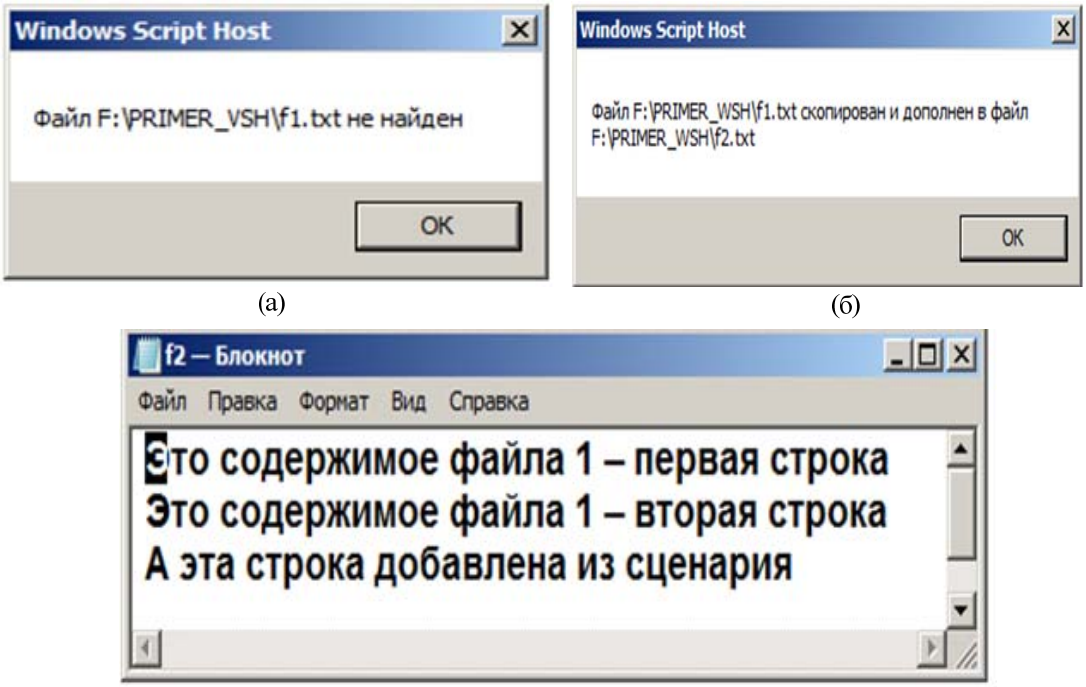


Рис. 2.19. Результат работы сценария Opwrfile.vbs:
а – при отсутствии файла-источника; б – диалоговое окно вывода сообщения о выполнении сценария; в – содержимое выходного файла, созданного в результате выполнения сценария

Приведенный ниже сценарий Appfile.vbs открывает файл fap.txt и добавляет в него текстовую строку «Строка добавлена сценарием Appfile». При многократном исполнении этого сценария в существующий файл каждый раз добавляется новая строка:

```
'Имя файла - Appfile.vbs
'Открыть файл fap.txt, добавить к его содержимому
'строку "Строка добавлена сценарием Appfile"
Const file1="F:\PRIMER_WSH\fap.txt" 'исходный текстовый файл
Const ForAppending =8
Dim fso ' Переменная-объект
Dim txtStream  'Объект TextStream
Set fso= CreateObject("Scripting.FileSystemObject") 'создали объект
If fso.FileExists (file1) Then    'проверили наличие файла - источника
  Set txtStream = fso.OpenTextFile (file1, ForAppending)
   'открыли файл для добавления строк
     txtStream.WriteLine "Строка добавлена сценарием Appfil
   Set txtStream=Nothing   ' освободить объект
  Wscript.Echo "Файл " & file1 & " существует, операция выполнена " & file2
Else
    Wscript.Echo "Файл " & file1 & " не найден "
End If
```

Результат работы сценария Appfile.vbs после двукратного его запуска представлен на рис. 2.20.

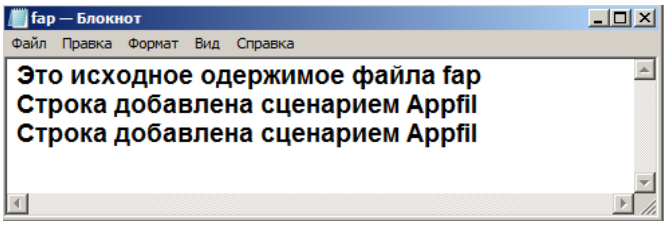


Рис. 2.20. Результат работы сценария AppFile.vbs
(двукратный запуск)

Представляет интерес решение задачи поиска и замены текста в файле.

В приведенном ниже сценарии, программа читает файл Fread.txt, находит фрагменты текста «s=» и заменяет их на «str=». Результат записывается в файл ftest.txt. При этом необходимо, чтобы файл сценария и обрабатываемые файлы находились в одной директории.

Для того чтобы прочесть файл Fread.txt, необходимо определить путь к этому файлу. Как было указано ранее, путь к сценарию определяется свойством объекта Wscript.ScriptFullName. Считывание этого свойства path= Wscript.ScriptFullName возвращает путь в виде выражения:

```
F:\PRIMER_WSH\Replfile.vbs.
```

Отделим от этого выражения имя сценария и получим путь к считываемому файлу. Такая задача легко решается с помощью функций VBS InStrRev(str1, str2), которая определяет номер последнего символа, с которого начинается вхождение подстроки str2 в строку str1, и Left(str, len), возвращающей len символов с начала строки str. Текст сценария, определяющего путь к считываемому файлу, реализованного в виде функции GetPath:

```
Function GetPath
   'Получить путь к сценарию
   Dim path
   path= Wscript.ScriptFullName
   GetPath=Left (path, InStrRev(path, "\"))
End Function
```

Используя этот путь, можно открыть входной файл и создать выходной файл следующей последовательностью команд:

```
path=GetPath() ' получить  текущий путь к сценарию
finpath=path&infile 'полное имя входного файла, где infile – имя вх. файла
foutpath=path&outfile 'полное имя выходного файла, где outfile – имя вых. файла
Set fso= CreateObject("Scripting.FileSystemObject") 'создать объект
```

```
'FileSystemObject для доступа к файловой системе
If  Not fso.FileExists (finpath) Then    'проверить наличие входного файла
    Wscript.Echo "Файл ' "& finpath & " ' не найден"
    Wscript.Quit 1 'завершить сценарий с кодом ошибки
End If
 Set txtStream = fso.OpenTextFile (finpath) 'открыть  исходный файл
 Set oFileOut= fso.OpenTextFile (foutpath, ForWriting, True) 'открыть/создать
    'выходной файл
```

Если файл существует, он будет открыт методом OpenTextFile, поскольку нужен лишь допуск к файлу для чтения, то в качестве параметра метода передается лишь имя файла. Для открытия выходного файла используется тот же метод, но с дополнительными параметрами ForWriting, позволяющий записывать информацию в файл, и True, определяющий возможность создания файла при его отсутствии.

Обработку входного файла и замену фрагментов текста, совпадающих с шаблоном, заданным фрагментом текста осуществляется следующей последовательностью команд:

```
Do While Not (txtStream.atEndOfStream) ' построчно читать входной файл
     Text=txtStream.ReadLine              ' читать строку
     Text = Filtstr (Text, pat, rep)      ' произвести замену в строке
     oFileOut.WriteLine Text              ' записать строку в выходной файл
  Loop
```

Если метод входного файла atEndOfStream возвращает True, то цикл Do While завершается. Если входной файл пуст, то метод сразу возвращает Thrue, при этом ни одна команда цикла не выполняется. В противном случае указанный метод возвращает Thrue только после того как метод ReadLine прочитает последнюю строку входного файла. Следующий оператор цикла вызывает пользовательскую функцию Filter, которая имеет три параметра: первый – Text, содержит подлежащую обработке строку; второй – pat, задает шаблон для поиска; третий – ret, задает фрагмент текста, который вставляется вместо шаблона. В примере параметры pat и ret заданы как константы. Последний оператор цикла записывает строку текста, которую возвращает функция Filtstr в выходной файл.

Команды, выполняющие замену текста, помещены в тело функции Filtstr:

```
Function Filtstr (txt, expr1, expr2)   ' заменить  в txt expr1 на expr2
  Dim oReg
  Set oReg= New RegExp                 ' создать регулярное  выражение
  oReg.Global=True                     ' обработать все совпадения
  oReg.IgnoreCase=True                 ' отключить чувствительность к регистру
```

```
  oReg.Pattern=expr1                    ' задать шаблон поиска
  Filtstr = oReg.Replace (txt, expr2)
End Function
```

VBS поддерживает объект «регулярное выражение», который создается с помощью выражения:

```
Set oReg= New RegExp.
```

Используя метод этого объекта Replace, можно обеспечить преобразование фрагментов строки по заданному шаблону. Свойство объекта Global, установленное в Thrue, позволяет обработать все фрагменты строки, совпадающие с шаблоном, а определение свойства объекта:

```
oReg.IgnoreCase=True
```

позволяет при поиске фрагментов, совпадающих с шаблоном, отключить чувствительность к регистру.

Полностью текст сценария, обеспечивающего обработку текстового файла, приведен ниже:

```
' Имя файла - Replfile.vbs
' Пример  -  открыть файл, искать фрагменты по шаблону,
' заменить их на заданную строку и записать результат в другой файл
' сценарий должен находиться в том же директории, что и файлы
Const infile="Fread.txt" 'исходный текстовый файл
Const outfile="ftest.txt" 'выходной текстовый файл
Const ForWriting =2 'режим записи
Const pat ="s=" 'шаблон для поиска
Const rep="str=" 'строка для замены
Dim Text '
Dim fso ' Переменная-объект
Dim txtStream,  oFileOut  'Объекты TextStream
Dim path, finpath, foutpath
path=GetPath() ' получить  текущий путь к сценарию
finpath=path&infile 'полное имя входного файла
foutpath=path&outfile 'полное имя выходного файла
Set fso= CreateObject("Scripting.FileSystemObject") 'создали объект
'FileSystemObject для доступа к файловой системе
If  Not fso.FileExists (finpath) Then    'проверили наличие входного файла
    Wscript.Echo "Файл ' "& finpath & " ' не найден"
    Wscript.Quit 1 'завершить сценарий с кодом ошибки
End If
Wscript.Echo "Файл:  " & finpath & vbCrLf&" будет записан в "&foutpath
   Set txtStream = fso.OpenTextFile (finpath) 'открыли  исходный файл
  Set oFileOut= fso.OpenTextFile (foutpath, ForWriting, True) 'открыли/создали
```

```
  'выходной файл
  Do While Not (txtStream.atEndOfStream)' построчно читать входной файл
     Text=txtStream.ReadLine  ' читать строку
     Text = Filtstr (Text, pat, rep)  ' произвести замену в строке
     oFileOut.WriteLine Text ' записать строку в выходной файл
   Loop
Wscript.Echo "Файл:  " & finpath & vbCrLf&" записан в "&foutpath
'*****используемые пользовательские функции**************
Function GetPath
   'Получить путь к сценарию
   Dim path
   path= Wscript.ScriptFullName
   Wscript.Echo("path="&path)
   GetPath=Left (path, InStrRev(path, "\"))
End Function
Function Filtstr (txt, expr1, expr2)
   ' заменить  в txt expr1 на expr2
  Dim oReg
  Set oReg= New RegExp ' создать регулярное  выражение
  oReg.Global=True ' обработать все совпадения
  oReg.IgnoreCase=True 'отключить чувствительность к регистру
  oReg.Pattern=expr1 'задать шаблон поиска
  Filtstr = oReg.Replace (txt, expr2)
End Function
```

Вопросы для самопроверки

1. Какие операции с объектами файловой системы могут быть выполнены с использованием объекта FileSystemObject?
2. Как создается объект FileSystemObject сценария WSH?
3. С какими файлами (тестовый, двоичный) позволяют работать методы объекта FileSystemObject?
4. Как прочитать содержимое файла с именем file, используя метод OpenTextFile?
5. Какие особенности следует учитывать при организации просмотра больших текстовых файлов с использованием метода Wscript.Echo и функции MsgBox()?
6. Как необходимо открыть файл file2.txt, чтобы он был доступен для записи в него информации?

3. ИСПОЛЬЗОВАНИЕ СЦЕНАРИЕВ ДЛЯ УПРАВЛЕНИЯ ОКНАМИ И ПРИЛОЖЕНИЯМИ

3.1. Запуск приложений Windows из сценария WSH

Для запуска приложений Windows из сценариев WSH можно использовать методы Run или Exec объекта WshShell. При использовании метода Run для запускаемого приложения можно задать тип окна (при условии, что приложение поддерживает этот тип). Так при выполнения следующего фрагмента 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.

Основные параметры метода Run представлены в табл. 3.1. Следует отметить, что метод Run всегда создает новый экземпляр запускаемого процесса. Кроме того, у метода Run существует еще необязательный параметр bWaitOnReturn является логической переменной, дающей указание ожидать завершения запущенного процесса. Если этот параметр не указан или установлен в false, то после запуска из сценария нового процесса управление сразу же возвращается обратно в сценарий (не дожидаясь завершения запущенного процесса). Если bWaitOnReturn установлен в true, то сценарий возобновит работу только после завершения вызванного процесса.

Приведенные ниже фрагменты сценария показывает особенности использования параметра bWaitOnReturn:

```
Set WshShell = WScript.CreateObject("WScript.Shell")
WshShell.Run "notepad", 1, true
Wscript.Echo "Управление передано  в сценарий".
```

При запуске указанного фрагмента сообщение «Управление передано в сценарий» появится на экране лишь после того как пользователь закончит работу с окном блокнота и закроет его. Использование Run без параметра bWaitOnReturn приведет к тому, что на экране появятся два окна, при этом на окно блокнота будет наложено окно сообщения.

Таблица 3.1

Перечень параметров метода Run

Значение	Константа VBS	Описание
0	vbHide	Прячет текущее окно и активизирует другое окно (показывает его и передает ему фокус)
1	vbNormalFocus	Активизирует и отображает окно. Если окно было минимизировано или максимизировано, система восстановит его первоначальное положение и размер. Этот флаг должен указываться сценарием во время первого отображения окна
2	vbMinimizedFocus	Активизирует окно и отображает его в минимизированном (свернутом) виде
3	vbMaximizedFocus	Активизирует окно и отображает его в максимизированном (развернутом) виде
4	vbNormalNoFocus	Отображает окно в том виде, в котором оно находилось последний раз. Активное окно при этом остается активным
5		Активизирует окно и отображает его в текущем состоянии
6	vbMinimizedNoFocus	Минимизирует заданное окно и активизирует следующее (в Z-порядке) окно
7		Отображает окно в свернутом виде. Активное окно при этом остается активным
8		Отображает окно в его текущем состоянии. Активное окно при этом остается активным
9		Активизирует и отображает окно. Если окно было минимизировано или максимизировано, система восстановит его первоначальное положение и размер
10		Устанавливает режим отображения, опирающийся на режим программы, которая запускает приложение

Следует обратить внимание на необходимость использования паузы в работе сценария для ожидания запуска приложения notepad:

```
Set WshShell = WScript.CreateObject("WScript.Shell")
WshShell.Run "notepad", 1
Wscript.Sleep 500 'пауза для синхронизации
Wscript.Echo "Управление передано  в сценарий"
```

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

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. Например:

```
Set WshShell = WScript.CreateObject("WScript.Shell")
        Set theNotepad = WshShell.Exec("notepad") 'создать ссылку на объект
Wscript.Sleep 500 'пауза для синхронизации
WScript.Echo  " Статус приложения: " & theNotepad.Status
theNotepad.Terminate '  закрыть запущенное ранее приложение
Wscript.Sleep 500 'пауза для синхронизации
WScript.Echo " Статус приложения: " & theNotepad.Status
```

Запуск указанного фрагмента сценария приводит к появлению окна приложения Блокнот, на которое накладывается активное окно метода Echo с сообщением о статусе приложения (рис. 3.1).

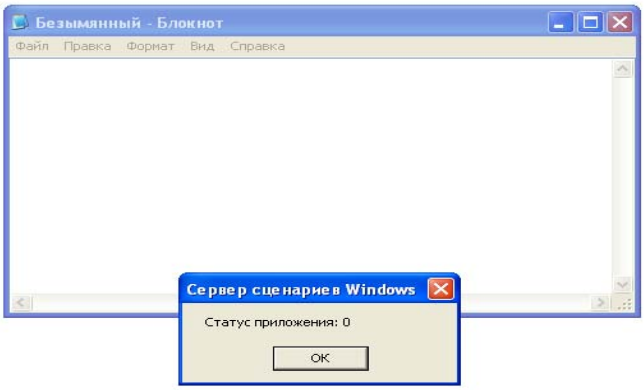


Рис. 3.1. Результаты работы фрагмента сценария
с запуском приложения Блокнот
и вывода на экран статуса приложения

Если нажать ОК, выполнение сценария продолжится. Выполняется метод Terminate, который завершает приложение и далее выводится информация о статусе этого приложения (статус приложения: 1).

При подобном запуске приложения в отличие от метода Run нельзя задать тип окна.

Вопросы для самопроверки

1. Какие методы объекта WshShell используются для запуска приложений Windows из сценария WSH?
2. Как из сценария запустить приложение «Блокнот» с использованием метода Run?
3. Какую информацию о приложении можно получить, используя свойство Status объекта WshScriptExec?
4. Как из сценария закрыть запущенное ранее приложение?

3.2 Синхронизация работы сценариев с другими процессами

Приостановить исполнение сценария позволяет метод Sleep объекта Wscript. Рассмотрим один из вариантов реализации синхронизации работы сценария с процессом считывания имени пользователя при входе в компьютерную сеть. Сценарий синхронизации можно реализовать с помощью следующей конструкции:

```
Set WshNetwork = WScript.CreateObject("Wscript.Network")   'создать объект
User=WshNetwork.UserName 'инициализировать значение
Do While User="" 'цикл - пока не получено имя пользователя
    Wscript.Sleep 200 'пауза для синхронизации
    User=WshNetwork.UserName 'считать свойство
Loop
Wscript.Echo  " имя пользователя  "& User 'печать имени.
```

Сценарий опрашивает в цикле свойство WshNetwork.UserName. Если свойство возвращает пустое значение, то сценарий ждет, когда оно станет непустым. У подобного опроса есть существенный недостаток: он загружает процессор на 100%. Поэтому в цикл добавлен оператор Wscript.Sleep 200, который приостанавливает исполнение сценария на 200 мс, в течение которых сценарий не занимает процессор.

Метод Sleep позволяет упорядочить работу с окнами. Пусть сценарий запускает два внешних приложения Calculator и Notepad, и выводит диалоговое окно для взаимодействия с пользователем:

```
Set WshShell = WScript.CreateObject("WScript.Shell") 'создаем объект
WshShell.Run "Calc.exe", 1 'запустить калькулятор
WshShell.Run "Notepad.exe", 1 'запустить блокнот
Wscript.Echo "Какое окно сверху ?" 'печать строки.
```

Можно предположить, что первым должно появиться окно Calculator, которое перекроется окном Notepad, а на переднем плане будет располагаться окно метода Echo. Однако при запуске сценария окна Notepad и Echo оказались закрыты окном Calculator, причем окно Echo располагается внизу на панели задач.

Это результат особенности обработки этих приложений в ОС Windows. Сценарий направляет оболочке два вызова Run и переходит к выполнению следующей команды. В результате окно Echo появится прежде, чем Windows запустит внешние приложения. Кроме того, порядок появления окон зависит от времени, необходимого для запуска конкретного приложения.

Чтобы установить порядок выведения окон на передний план, необходимо последовательно открывать окна приложений, предусмотрев в сценарии ожидание появления этих окон. Это можно реализовать, добавив небольшую паузу после обращения к методу Run:

```
WshShell = WScript.CreateObject("WScript.Shell") 'создаем объект
WshShell.Run "Calc.exe", 1 'запустить калькулятор
Wscript.Sleep 500 'ожидать 0,5 сек
WshShell.Run "Notepad.exe", 1 'запустить блокнот
Wscript.Sleep 500 'ожидать 0,5 сек
Wscript.Echo "Какое окно сверху?" 'печать строки
```

Вызов метода Sleep переводит сценарий в состояние паузы на 0,5 с перед обработкой очередной команды. В результате окна приложений и окно Echo будут появляться на экране в заданном порядке.

Кроме метода Run, активизировать окно приложения можно с помощью метода obj.AppActivate title. Переменная-объект obj содержит ссылку на объект WshShell, а title – текст заголовка окна уже запущенного приложения, на которое нужно перевести фокус.

Ниже приведен сценарий Async2.vbs, в котором запускаются два приложения – Notepad и Calculator. После запуска сценарий запрашивает у пользователя заголовок окна, которое необходимо активизировать, и переводит фокус на окно, заголовок которого совпадает с введенным текстом:

```
'Имя файла - Async2.vbs
'Переключение между окнами методом AppActivate
Const Edit_Tit = "Безымянный–Блокнот" 'заголовок окна
Const Calc_Tit = "Калькулятор" 'заголовок окна
Dim Wsh, win_title
Set WshShell = WScript.CreateObject("WScript.Shell") 'создаем объект
  WshShell.Run "Calc.exe", 1 'запустить калькулятор
  Wscript.Sleep 500 'ожидать 0,5 сек 'передать фокус на калькулятор
  WshShell.Run "Notepad.exe", 1 'запустить блокнот
  Wscript.Sleep 500 'ожидать 0,5 сек 'передать фокус на блокнот
  win_title = InputBox("Введите имя окна на которое передать фокус",  _
                       "Для ввода наименования окна", Calc_tit)
   WshShell.AppActivate win_title' перевод фокуса на определенное окно
   Wscript.Sleep 3000 'ожидать 3 сек, контролировать фокус на калькуляторе
win_title = InputBox("Введите имя окна на которое передать фокус",  _
                     "Для ввода наименования окна", Edit_tit)
   WshShell.AppActivate win_title' перевод фокуса на определенное окно
```

Метод AppActivate обладает рядом недостатков, которые затрудняют его использование. В частности необходимо знать заголовок активируемого окна (наименование окна зависит от локализации ОС); если программа (приложение) имеет возможность изменять наименование окна, то использование метода существенно затрудняется; если несколько окон имеют одинаковые заголовки, то метод будет активизировать первый экземпляр окна (по внутреннему списку открытых окон); нельзя развернуть свернутое окно.

Вопросы для самопроверки

1. Каким образом обеспечить паузу в исполнении сценария, необходимую для завершения операции запуска приложения?

2. Какой метод объекта WshShell позволяет управлять переводом «фокуса» с одного открытого ранее окна приложения на другое?

3. Назовите недостатки метода AppActivate?

4. Какую задержку выполнения сценария (в секундах) обеспечивает метод Wscript.Sleep 750?

3.3. Имитация нажатия клавиш

Производить переключение между окнами нескольких запущенных приложений позволяет рассмотренный выше 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 указаны в разд. 3.1.

Из сценария возможно запустить приложение и перевести его окно на передний план (сделать активным). В этом случае все нажатия клавиш пользователем передаются в активное (на переднем плане) окно. Представляется достаточно важным возможность имитировать в сценарии нажатия клавиш пользователем. Для этого используется метод SendKeys объекта WshShell. Синтаксис использования метода:

```
WshShell.SendKeys keys,
```

где параметр keys содержит один или несколько передаваемых в активное окно символов.

Так, чтобы имитировать нажатие клавиши A, в параметр keys необходимо передать символ "A". Если необходимо передать нажатие нескольких клавиш, то в параметр передаются несколько символов ("ABC").

Некоторые символы имеют в методе SendKeys специальное значение: +, ^, %, ~, (,). Для того чтобы задать один из этих символов, их нужно заключить в фигурные скобки {}. Например, для задания знака плюс используется {+}. Квадратные скобки [] хотя и не имеют в методе SendKeys специального смысла, их также нужно заключать в фигурные скобки. Кроме этого, для задания самих фигурных скобок следует использовать следующие конструкции: {{} (левая скобка) и {}} (правая скобка).

Для задания неотображаемых символов, таких как <Enter> или <Tab>, и специальных клавиш в методе SendKeys используются коды, представленные в табл. 3.2.

Таблица 3.2

Коды специальных клавиш метода SendKeys

Наименование клавиши	Код	Наименование клавиши	Код
<Backspace>	{BACKSPACE}, {BS}{BKSP}	<→ >	{RIGHT}
<Break>	{BREAK}	<F1>	{F1}
<Caps Lock>	{CAPSLOCK}	<F2>	{F2}
<Del> или <Delete>	{DELETE} {DEL}	<F3>	{F3}
<End>	{END}	<F4>	{F4}
<Enter>	{ENTER} или ~	<F5>	{F5}
<Esc>	{ESC}	<F6>	{F6}
<Home>	{HOME}	<F7>	{F7}
<Ins> или <Insert>	{INSERT} или {INS}	<F8>	{F8}
<Num Lock>	{NUMLOCK}	<F9>	{F9}
<Page Down>	{PGDN}	<F10>	{F10}
<Page Up>	{PGUP}	<F11>	{F11}
<Print Screen>	{PRTSC}	<F12>	{F12}
<Scroll Lock>	{SCROLLLOCK}	<F13>	{F13}
<Tab>	{TAB}	<F14>	{F14}
<↑>	{UP}	<F15>	{F15}
<← >	{LEFT}	<F16>	{F16}
<↓>	{DOWN}		

Для задания комбинаций клавиш с <Shift>, <Ctrl> или <Alt> перед соответствующей клавишей нужно поставить один или несколько кодов: для <Shift> – код "+", для <Ctrl> – код "^" и для <Alt> – код "%".

Для того чтобы задать комбинацию клавиш, которую нужно набирать, удерживая нажатыми клавиши <Shift>, <Ctrl> или <Alt>, нужно заключить коды этих клавиш в скобки. Например, если требуется сымитировать нажатие клавиш G и S при нажатой клавише <Shift>, следует использовать последовательность "+(GS)". Для того чтобы задать одновременное нажатие клавиш <Shift>+<G>, а затем <S> (уже без <Shift>), используется "+GS".

В методе SendKeys можно задать несколько нажатий подряд одной и той же клавиши. Для этого необходимо в фигурных скобках указать код нужной клавиши, а через пробел – число нажатий. Например, {A 42} означает нажатие клавиши <A> 42 раза подряд; {h 10} означает нажатие клавиши <h> 10 раз подряд.

Рассмотрим пример использования метода SendKeys с двумя приложениями:

```
Set WshShell = WScript.CreateObject("WScript.Shell")
WScript.Echo("Запускаем калькулятор и" & vbCrLf & "считаем 10+5")
' Создаем объект WshScriptExec (ссылка theCalculator)
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.
```

Приведенный фрагмент сценария запускает калькулятор и делает его окно активным. Далее имитируем нажатие клавиш калькулятора, которые реализуют операцию "10 + 5 =". Посылаем нажатия клавиш в окно Калькулятора:

```
WshShell.SendKeys "10"
WshShell.SendKeys "{+}"
WshShell.SendKeys "5"
WshShell.SendKeys "="
WScript.Sleep 500.
```

Далее копируем результат вычисления в буфер, выводим на экран сообщение и закрываем окно калькулятора. Прежде чем закрыть окно калькулятора его снова необходимо активизировать:

```
' 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}".
```

После окончания работы калькулятора запускаем Блокнот, активизируем окно и имитируем нажатие клавиш 10 + 5 =. Эта информация должна появиться в окне Блокнота. Затем вставляем информацию из буфера (имитируем нажатие клавиш <Ctrl>+V) и выводим на экран текстовую информацию:

```
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 "10{+}5="
' Вставляем содержимое буфера Windows (<Ctrl>+V)
```

```
WshShell.SendKeys "^v"
' Выводим в окно Блокнота оставшуюся информацию
WshShell.SendKeys " {(}c{)} Calculator {+} Notepad".
```

Следует отметить, что буквы кириллицы корректно в окно не передаются. Полностью текст сценария Calc_not.vbs приведен ниже:

```
'Имя файла - Calc_not.vbs
'Имитация нажатия клавиш
Option Explicit
Dim WshShell, theCalculator  ' Объявляем переменные
' Создаем объект WshShell
Set WshShell = WScript.CreateObject("WScript.Shell")
WScript.Echo("Запускаем калькулятор и" & vbCrLf & "считаем 10+5")
' Создаем объект WshScript (запускаем Калькулятор)
Set theCalculator = WshShell.Exec("calc")
' Приостанавливаем сценарий до прявления окна Калькулятора
WScript.Sleep 500
' Активизируем окно Калькулятора
WshShell.AppActivate theCalculator.ProcessID
' Посылаем нажатия клавиш в окно Калькулятора
WshShell.SendKeys "10"
WScript.Sleep 700
WshShell.SendKeys "{+}"
WScript.Sleep 700
WshShell.SendKeys "5"
WScript.Sleep 700
WshShell.SendKeys "="
WScript.Sleep 700
' 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 сценарий до прявления окна Блокнота
WScript.Sleep 1000
WshShell.AppActivate "notepad"    ' Активизируем окно Блокнота
' Посылаем нажатия клавиш в окно Блокнота
WshShell.SendKeys "10"
```

```
WScript.Sleep 700
WshShell.SendKeys "{+}"
WScript.Sleep 700
WshShell.SendKeys "5"
WScript.Sleep 700
WshShell.SendKeys "="
WScript.Sleep 700
' Вставляем содержимое буфера Windows (<Ctrl>+V)
WshShell.SendKeys "^v"
' Выводим в окно Блокнота оставшуюся информацию
WshShell.SendKeys " {(}c{)} Calculator {+} Notepad"
```

Методы Sleep между командами сценария добавлены для повышения наглядности работы сценария.

Вопросы для самопроверки

1. Как можно получить информацию об идентификаторе запущенного из сценария процесса?
2. Какой метод позволяет имитировать в сценарии нажатие клавиш пользователем?
3. Какие символы (клавиши) необходимо заключать в фигурные скобки при использовании их в качестве параметров метода SendKeys?
4. Последовательность нажатия каких клавиш передает следующая комбинация WshShell.SendKeys "+GS"?

3.4. Обработка ошибок при исполнении сценариев WSH

Режим обработки ошибок (исключений) при исполнении сценариев в VBScript включается с помощью оператора On Error Resume Next. Если после этого при исполнении какого-либо оператора в сценарии произойдет ошибка, то управление передастся к следующему оператору в тексте.

Здесь рассматриваются ошибки, возникающие в ходе работы сценария, это не те ошибки, когда неправильно поставлены скобки, запятые или ошибку содержит имя переменной. Это ошибки, возникающие в некоторых (исключительных) ситуациях во время выполнения сценария.

Для анализа ошибок используется специальный объект Err, который содержит два свойства: Number – числовой код возникшей ошибки и Description – краткое описание этой ошибки.

Оператор On Error GoTo 0 отменяет обработку ошибок при исполнении сценариев. После исполнения этого оператора, текст сценария продолжает обрабатываться обработчиком сценариев.

Следует отметить, что включение режима обработки ошибок при исполнении сценариев подавляет все сообщения о синтаксических ошибках обработчика сценариев. Поэтому ответственность за обнаружение ошибок и оповещение об их присутствии ложится на выполняемый сценарий.

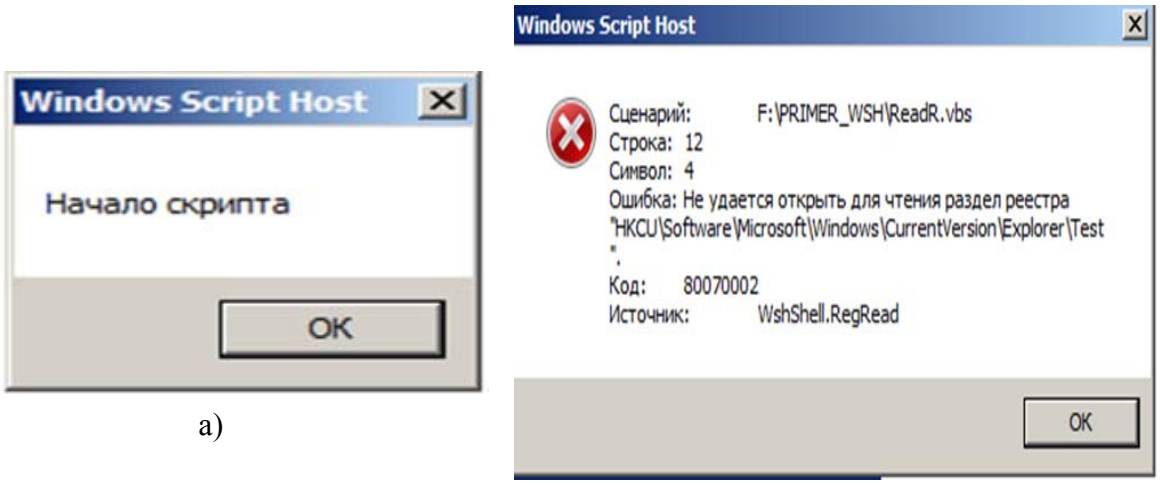
В этом случае, если в сценарии будет находиться оператор, содержащий ошибку, то он не будет выполнен, а обработчик сценариев перейдет к следующему оператору.

Чтобы не допускать таких ситуаций, необходимо использовать On Error GoTo 0, всякий раз, как только отпадает необходимость в использовании On Error Resume Next.

Рассмотрим пример сценария ReadR.vbs, пытающегося прочитать значение некоторого параметра в реестре и вывести его значение на экран:

```
'Имя файла - ReadR.vbs
Dim Perem
set WSHShell = WScript.CreateObject("WScript.Shell")
WSHShell.Popup "Начало скрипта"
Perem = ""
Perem =
WSHShell.RegRead("HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Test")
WSHShell.Popup Perem
WSHShell.Popup "Конец скрипта"
```

Результаты работы сценария ReadR.vbs представлены на рис. 3.2.



а) б)

Рис. 3.2. Результаты работы сценария ReadR.vbs:
а – первый этап работы сценария; б – заключительный этап работы сценария

Поскольку появилось сообщение «Начало скрипта», то синтаксических ошибок текст сценария не содержит, но затем появляется сообщение об ошибке. Это объясняется тем, что параметра с именем Test в разделе HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer не существует. Из-за этого пример не отработал до конца.

Для устранения таких ситуаций в следующем примере воспользуемся оператором On Error Resume Next и последующим анализом возможной ошибки считывания параметра реестра:

```
'Имя файла - ReadRE.vbs
Dim Perem
set WSHShell = WScript.CreateObject("WScript.Shell")
WSHShell.Popup "Начало скрипта"
Perem = ""
On Error Resume Next
GetErr()
If Err.Source="WshShell.RegRead" Then
Perem = "Параметр не существует"
Else
Perem =
WSHShell.RegRead("HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Test")
End If
Err.Clear
On Error GoTo 0
WSHShell.Popup Perem
WSHShell.Popup "Конец скрипта"
Function GetErr()
If
WSHShell.RegRead("HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Test")
Then
End If
End Function
```

Попытка прочитать параметр из реестра вынесена в отдельную функцию. Если происходит ошибка – функция возвращает код ошибки (Err.Number) и источник (Err.Source). В операторе if проверяется тип ошибки и если это именно ошибка чтения параметра реестра, то значит параметр не удалось прочитать по причине его отсутствия, о чем и выводится на экран сообщение. При этом сценарий корректно обрабатывается до конца. Если ошибка не возникает, то значение параметра считывается в переменную и выводится на экран.

Возможна программная имитация ошибок исполнения сценария. Для этого используется метод Raise объекта Err. Ниже приведен сценарий, который генерирует ошибку с кодом 5 и выводит на экран ее описание:

```
'Имя файла - GenEr.vbs
On Error Resume Next
GetErr()
Err.Clear
```

```
Err.Raise 5 'генерация ошибки
MsgBox "Код ошибки:  " & Err.Number &vbCrLf & " Описание ошибки : " & Err.Description
 On Error GoTo 0
```

Результат работы сценария представлены на рис. 3.3.

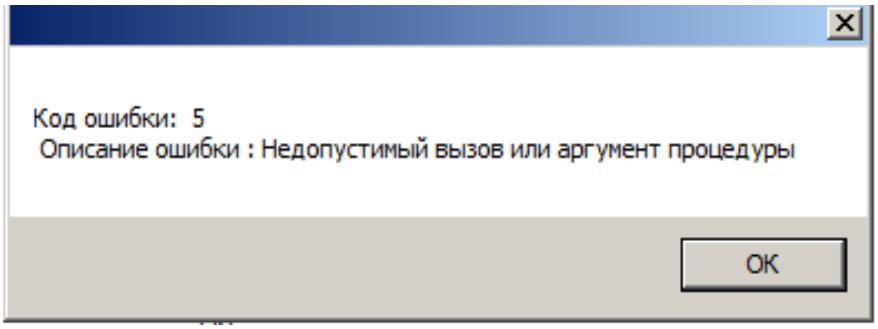


Рис. 3.3. Результаты работы сценария GenEr.vbs

На основе этого примера можно составить сценарий, который запрашивает код ошибки и возвращает описание этой ошибки.

Вопросы для самопроверки

1. Выполнение какого оператора включает режим обработки ошибок сценария?
2. Какой оператор отменяет обработку ошибок сценария?
3. Какой объект используется для анализа ошибок сценария?
4. Какой метод объекта Err используется для программной имитации ошибок?

4. ИСПОЛЬЗОВАНИЕ СЦЕНАРИЕВ ДЛЯ АДМИНИСТРИРОВАНИЯ КОМПЬЮТЕРНЫХ СИСТЕМ

4.1. Работа с системным реестром Windows XP

Системный реестр операционной системы – это база данных, в которой хранится информация о конфигурации компьютера, операционной системы и инсталлированных на компьютере программ. С точки зрения пользователя, реестр является иерархическим деревом разделов, подразделов и параметров. Работать с этим деревом можно с помощью стандартного редактора реестра regedit.exe, встроенного в Windows.

WSH предоставляет весьма удобный способ манипулирования реестром с помощью методов RegRead, RegWrite и RegDelete объекта WshShell. В Windows XP для работы с системным реестром сценарий должен иметь разрешение на доступ к разделам реестра, которым обладает администратор.

Ниже рассматривается сценарий RegVsh.vbs, который производит манипуляции внутри корневого раздела HKEY_CURRENT_USER, причем каждая операция выполняется только после утвердительного ответа на соответствующий запрос, который формируется в диалоговом окне.

Сначала в разделе HKEY_CURRENT_USER создается подраздел NwpiKey, в который затем записывается строковый параметр ExampleNwpi со значением "Nwpi from WSH". Ниже приводится пример программы RegVsh.vbs работы с системным регистром, которая включает все необходимые для ее понимания комментарии:

```
'Имя файла - RegVsh.vbs
Option Explicit
Dim WshShell,Root,Key,Res,SValue,ValueName,SRegValue
Root="HKEY_CURRENT_USER"   'Корневой ключ
Key="\NwpiKey\"         'Новый ключ
ValueName="ExampleNwpi"   'Имя нового параметра
SValue="Nwpi from WSH"    'Значение нового параметра
'Создаем объект WshShell
Set WshShell=WScript.CreateObject("WScript.Shell")
'Запрос на создание нового ключа
Res=WshShell.Popup("Создать ключ" & vbCrLf & Root & Key & "?",0,_
 "Работа с реестром",vbQuestion+vbYesNo)
If Res=vbYes Then   'Нажата кнопка Да
 'Записываем новый ключ
 WshShell.RegWrite Root & Key, ""
 WshShell.Popup "Ключ" & vbCrLf & Root & Key & " создан!",0,_
  "Работа с реестром",vbInformation+vbOkOnly
End If
```

```
'Запрос на запись нового параметра
Res=WshShell.Popup("Записать параметр" & vbCrLf & Root & Key & _
 ValueName & "?",0,"Работа с реестром",vbQuestion+vbYesNo)
If Res=vbYes Then   'Нажата кнопка Да
 'Записываем новый строковый параметр
 WshShell.RegWrite Root & Key & ValueName,SValue,"REG_SZ"
 WshShell.Popup "Параметр" & vbCrLf & Root & Key & _
  ValueName & " записан!",0,"Работа с реестром",vbInformation+vbOkOnly
 'Считываем значение созданного параметра
 SRegValue=WshShell.RegRead(Root & Key & ValueName)
 'Выводим на экран полученное значение
 WshShell.Popup Root & Key & ValueName & "=" & SRegValue,0,_
  "Работа с реестром",vbInformation+vbOkOnly
End If
'Запрос на удаление параметра
Res=WshShell.Popup("Удалить параметр" & vbCrLf & Root & Key & _
 ValueName & "?",0,"Работа с реестром",vbQuestion+vbYesNo)
If Res=vbYes Then   'Нажата кнопка Да
 'Удаляем параметр
 WshShell.RegDelete Root & Key & ValueName
 WshShell.Popup "Параметр" & vbCrLf & Root & Key & _
  ValueName & " удален!",0,"Работа с реестром",vbInformation+vbOkOnly
End If
'Запрос на удаление раздела
Res=WshShell.Popup("Удалить раздел" & vbCrLf & Root & Key & _
 "?",0,"Работа с реестром",vbQuestion+vbYesNo)
If Res=vbYes Then   'Нажата кнопка Да
 'Удаляем раздел
 WshShell.RegDelete Root & Key
 WshShell.Popup "Раздел" & vbCrLf & Root & Key & " удален!",0,_
  "Работа с реестром",vbInformation+vbOkOnly
End If
```

Вопросы для самопроверки

1. Какая информация хранится в системном реестре ОС?
2. Методы какого объекта позволяют работать с параметрами системного реестра?
3. Для чего служат ключи в структуре системного реестра?
4. Какой метод объекта WshShell позволяет удалить выбранный параметр системного реестра?

4.2. Работа с ресурсами локальной компьютерной сети

Основные возможности работы с сетью можно реализовать при помощи объекта WScript.Network. Создание этого объекта осуществляется следующим образом:

```
Set WshNetwork = WScript.CreateObject("WScript.Network").
```

Объект обладает тремя свойствами и восемью методами. Свойства ComputerName, UserDomain и UserName возвращают соответственно имя компьютера, имя домена и имя текущего пользователя.

Определение имен рабочей станции, пользователя и домена

Рассмотрим сценарий NetPar.vbs, определяющий имя машины, пользователя и домена (рис. 4.1):

```
'Имя файла - NetPar.vbs
Option Explicit
Dim WshNetwork,s,NetwDrives,i,NetwPrinters  ' Объявляем переменные
' Создаем объект WshNetwork
Set WshNetwork = WScript.CreateObject("WScript.Network")
s="Сетевые параметры станции:" & vbCrLf & vbCrLf
' Выводим на экран свойства ComputerName, UserName и UserDomain
s=s & "Имя машины: " & WshNetwork.ComputerName & vbCrLf
s= s & "Имя пользователя: " & WshNetwork.UserName & vbCrLf
s= s & "Домен: " & WshNetwork.UserDomain
WScript.Echo s
```

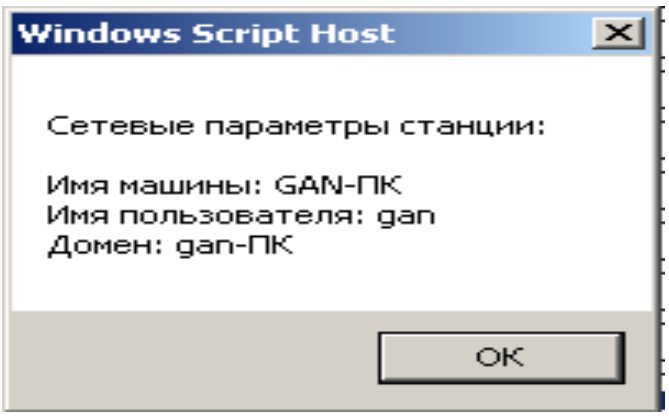


Рис. 4.1. Результаты работы сценария NetPar.vbs

Получение списка подключенных сетевых дисков и принтеров

У объекта WshNetwork имеются методы EnumNetworkDrives и EnumPrinterConnections, с помощью которых можно создать коллекции, содержащие соответственно сведения обо всех подключенных к локальной

станции сетевых дисках и сетевых принтерах. Эти коллекции устроены следующим образом: первым элементом является буква диска или название порта, вторым – сетевое имя ресурса, с которым связан этот диск или принтер. Та же последовательность сохраняется для всех элементов коллекции.

Ниже представлен сценарий, формирующий список сетевых дисков и принтеров:

```
'Имя файла - NetRes.vbs
Option Explicit
Dim WshNetwork,s,NetwDrives,i,NetwPrinters  ' Объявляем переменные
' Создаем объект WshNetwork
Set WshNetwork = WScript.CreateObject("WScript.Network")
'********   Вывод списка всех подключенных сетевых дисков   *********
s="Подключенные сетевые диски:" & vbCrLf & vbCrLf
' Создаем коллекцию с данными о подключенных дисках
Set NetwDrives = WshNetwork.EnumNetworkDrives()
i=0
While i<=NetwDrives.Count()-2  ' Перебираем элементы коллекции
 ' В первом элементе коллекции содержится буква диска,
 ' во втором - сетевое имя ресурса и т.д.
 s=s & NetwDrives.Item(i) & "  " & NetwDrives.Item(i+1) & vbCrLf
 i=i+2
Wend
WScript.Echo s   ' Выводим сформированные строки на экран
'********    Вывод списка всех подключенных сетевых принтеров    *******
s="Подключенные сетевые принтеры:" & vbCrLf & vbCrLf
' Создаем коллекцию с данными о подключенных принтерах
Set NetwPrinters = WshNetwork.EnumPrinterConnections()
i=0
While i<=NetwPrinters.Count()-2   ' Перебираем элементы коллекции
 ' В первом элементе коллекции содержится названия локальных портов,
 ' во втором - сетевое имя принтера и т.д.
 s=s & NetwPrinters.Item(i) & "  " & NetwPrinters.Item(i+1) & vbCrLf
 i=i+2
Wend
WScript.Echo s  'Выводим сформированные строки на экран
```

Подключение и отключение сетевых дисков и принтеров

Имеющиеся в локальной сети общедоступные ресурсы (диски и принтеры) можно посредством сценария подключить к рабочей станции для совместного использования. Подключаемому сетевому диску при этом нужно поставить в соответствие незанятую букву локального диска. Например, если в системе уже имеются диски: C:, D: и E: (локальные или сетевые), то

сетевой диск можно подключить под буквой F: или K:, но не E:. В случае подключения сетевого принтера можно напрямую соединиться с этим принтером для печати из приложений Windows или поставить в соответствие удаленному принтеру локальный порт (для печати из старых приложений MS-DOS).

В качестве примера рассмотрим сценарий SwRes.vbs, в котором производится подключение диска K: к сетевому ресурсу \\RS_NT_Server\d и установка связи локального порта LPT1 с сетевым принтером \\104_Stepankova\HP.

Сначала нужно создать экземпляры объектов WshNetwork и WshShell:

```
Set WshNetwork = WScript.CreateObject("WScript.Network"),
Set WshShell = WScript.CreateObject("WScript.Shell").
```

Для того чтобы подключить сетевой диск к устройству "K:", нужно быть уверенным, что с этой буквой уже не связан сетевой диск (иначе произойдет ошибка). Поэтому предварительно отключается сетевой диск с помощью метода RemoveNetworkDrive:

```
WshNetwork.RemoveNetworkDrive Drive,
```

переменной Drive заранее было присвоено значение "K:". При выполнении этой команды может произойти ошибка (например, диск "K:" не существует или возникла ошибка при отключении связанного с ним сетевого ресурса), поэтому следует обработать эту ситуацию следующим образом:

```
If Err.Number<>0 Then
Mess="Ошибка при отключении диска " & Drive & vbCrLf & _
 "Код ошибки: " &  e.number & vbCrLf &+ _
 "Описание: " & e.description
WshShell.Popup Mess,0,"Отключение сетевого диска",vbCritical
Else
' Все в порядке
Mess="Диск " & Drive & " отключен успешно"
WshShell.Popup Mess,0,"Отключение сетевого диска",vbInformation
End If
```

Теперь в случае возникновения ошибки при работе метода RemoveNetworkDrive в полях Number и Description переменной-объекта Err будут соответственно содержаться код и описание ошибки. Если отключение диска "K:" прошло успешно, на экран выводится соответствующее диалоговое окно с информацией об этом.

Аналогично обрабатываем ошибки при подключении сетевого диска:

```
If Err.Number<>0 Then
Mess="Ошибка при подключении диска " & Drive & " к " & NetPath &_
 "Код ошибки: " & e.number & "Описание: " & e.description
```

```
 WshShell.Popup Mess,0,"Подключение сетевого диска",vbCritical
Else
 ' Все в порядке
 Mess="Диск " & Drive & " успешно подключен к " & NetPath
 WshShell.Popup Mess,0,"Подключение сетевого диска",vbInformation
End If.
```

Освобождение локального порта (метод RemovePrinterConnection), подключение сетевого принтера к этому порту (метод AddPrinterConnection) и обработка ошибок времени выполнения, которые могут возникнуть при этих действиях, производится в сценарии аналогичным образом.

Реализация сценария MapResources.vbs представлена ниже:

```
' Имя: MapResources.vbs
' Описание: Отключение и подключение сетевых дисков и принтеров
Option Explicit
' Объявляем переменные
Dim WshNetwork,Drive,NetPath,Port,NetPrinter
Drive="K:"   ' Буква диска
NetPath="\\RS_NT_Server\d"    ' Сетевой путь для подключения диска
Port="LPT1"   ' Название локального порта
' Сетевой путь для подключения принтера
NetPrinter="\\104_Stepankova\HP"
' Создаем объект WshNetwork
Set WshNetwork = WScript.CreateObject("WScript.Network")
' Создаем объект WshShell
Set WshShell = WScript.CreateObject("WScript.Shell")
On Error Resume Next  ' Включаем обработку ошибок времени выполнения
'*************  Отключение сетевого диска  ***********************
' Отключаем сетевой диск
WshNetwork.RemoveNetworkDrive Drive
If Err.Number<>0 Then
 Mess="Ошибка при отключении диска " & Drive & vbCrLf & _
  "Код ошибки: " &  e.number & vbCrLf &+ _
  "Описание: " & e.description
 WshShell.Popup Mess,0,"Отключение сетевого диска",vbCritical
Else
 ' Все в порядке
 Mess="Диск " & Drive & " отключен успешно"
 WshShell.Popup Mess,0,"Отключение сетевого диска",vbInformation
End If
'*************  Подключение сетевого диска  ***********************
' Подключаем сетевой диск
WshNetwork.MapNetworkDrive Drive,NetPath
If Err.Number<>0 Then
```

```
 Mess="Ошибка при подключении диска " & Drive & " к " & NetPath &_
  "Код ошибки: " & e.number & "Описание: " & e.description
 WshShell.Popup Mess,0,"Подключение сетевого диска",vbCritical
Else
 ' Все в порядке
 Mess="Диск " & Drive & " успешно подключен к " & NetPath
 WshShell.Popup Mess,0,"Подключение сетевого диска",vbInformation
End If
'*************  Освобождение локального порта  **********************
' Разрываем связь с сетевым принтером
WshNetwork.RemovePrinterConnection Port
If Err.Number<>0 Then
 Mess="Ошибка при отключении порта " & Port & "Код ошибки: " &_
 e.number & "Описание: " & e.description
 WshShell.Popup Mess,0,"Отключение порта от сетевого ресурса",vbCritical
Else
 ' Все в порядке
 Mess="Порт " & Port & " отключен успешно"
 WshShell.Popup Mess,0,"Отключение порта от сетевого ресурса",_
  vbInformation
End If
'*****  Подключение локального порта  к сетевому принтеру  *********
' Подключаем сетевой принтер к локальному порту
WshNetwork.AddPrinterConnection Port,NetPrinter
If Err.Number<>0 Then
 Mess="Ошибка при переназначении порта " & Port & " на " & NetPrinter &_
  "Код ошибки: " & e.number & "Описание: " & e.description
 WshShell.Popup Mess,0,"Подключение порта к сетевому ресурсу",vbCritical
Else
 ' Все в порядке
 Mess="Порт " & Port & " успешно подключен к " & NetPrinter
 WshShell.Popup Mess,0,"Подключение порта к сетевому ресурсу",
  vbInformation
End If.
```

Для решения более сложных задач, связанных с администрированием локальной сети, можно применять имеющиеся в Windows XP технологии ADSI – Active Directory Service Interface и WMI – Windows Management Instrumentation.

Вопросы для самопроверки

1. Какой объект позволяет работать с ресурсами компьютерной сети?
2. Как определить имя домена, в котором находится компьютер?
3. Как подключить системный диск к выбранному устройству?
4. Какой метод объекта WshNetwork используется для отключения системного диска?

4.3. Запуск сценариев на удаленных машинах

Начиная с версии 5.6 сценарии WSH можно запускать не только на локальной машине, но и на других компьютерах, имеющихся в сети (это может быть очень удобно для централизованного администрирования удаленных рабочих станций).

Такие WSH-сценарии называются удаленными сценариями (remote scripts). При этом файл со сценарием может находиться либо на локальной машине, либо на общедоступном сетевом ресурсе. На жесткий диск удаленной машины файл сценария копироваться не будет – вместо этого текст сценария по коммуникационному протоколу DCOM – Distributed COM (распределенный COM) передается непосредственно в память процесса, запускаемого на этой машине.

Естественно, такой механизм запуска сценариев является весьма мощным средством удаленного администрирования, однако он также может быть использован для быстрого и практически незаметного распространения по сети вирусов. Поэтому при использовании удаленных сценариев WSH предусмотрены довольно строгие меры безопасности.

Во-первых, и на локальной, и на удаленной машинах должны быть установлены операционные системы Windows NT (SP3)/Windows 2000/ Windows XP (системы Windows 95/98/ME не поддерживаются).

Во-вторых, пользователь, который запускает сценарий, должен входить в группу локальных администраторов на той машине, где должен выполняться сценарий.

В-третьих, удаленная машина должна быть предварительно настроена для выполнения удаленных сценариев (по умолчанию после первоначальной установки выполнение таких сценариев запрещено). Для этого необходимо записать 1 в следующий параметр системного реестра:

HKLM\Software\Microsoft\Windows Script Host\Settings\Remote. (1)

Причем, если этот параметр не существует, его нужно создать. Если значением этого параметра является 0, то это означает, что выполнение удаленных сценариев на машине запрещено. Для того чтобы разрешить выполнение удаленных сценариев на уровне пользователя, необходимо создать параметр:

HKCU\Software\Microsoft\Windows Script Host\Settings\Remote (2)

и записать в него 1. Если значением этого параметра является 0, то это означает, что выполнение удаленных сценариев для текущего пользователя запрещено.

Также при настройке режима выполнения удаленных сценариев нужно проверить значение параметра:

HKLM\Software\Microsoft\Windows Script Host\Settings\IgnoreUserSettings. (3)

Если значением этого параметра является 1, то параметр (2) игнорируется и проверяется только значение параметра (1). Если значением параметра (3) является 0, то WSH сначала проверяет параметр (2) и только в случае его отсутствия принимается во внимание значение параметра (1).

Если удаленные сценарии нужно выполнять на машине с операционной системой Windows XP, то на этой машине нужно перерегистрировать сервер wscript.exe с помощью следующей команды:

```
wscript.exe–regserver.
```

Удаленные сценарии всегда запускаются с помощью сервера wscript.exe, причем в этих сценариях не поддерживается вывод на экран удаленного компьютера никаких элементов пользовательского интерфейса (не выводятся даже диалоговые окна с сообщениями о возникающих в ходе выполнения ошибках). Другими словами, в удаленных сценариях по умолчанию нельзя использовать методы WScript.Echo или WshShell.Popup (это может привести к непредсказуемым результатам).

Для примера рассмотрим сценарий RemoteShortcut.vbs, который создает ярлык к калькулятору и помещает его в папку Автозагрузка. Предположим, что этот сценарий находится в корневом каталоге диска "D:", а запустить сценарий необходимо на компьютере "\\Stand":

```
'Имя файла - RemoteShortcut.vbs
' Создаем ярлык
 Dim WSHShell
Set WSHShell = WScript.CreateObject("WScript.Shell")
 Dim MyShortcut, MyDesktop, StartupPath
 ' Узнаем путь к специальной папке Автозагрузка
StartupPath = WSHShell.SpecialFolders("Startup")
 ' Создаем ярлык для Автозагрузки
Set MyShortcut = WSHShell.CreateShortcut(StartupPath & _
"\Ярлык к калькулятору.lnk")
 ' Устанавливаем свойства для ярлыка
MyShortcut.TargetPath =
WSHShell.ExpandEnvironmentStrings("%windir%\calc.exe")
MyShortcut.WorkingDirectory = WSHShell.ExpandEnvironmentStrings("%windir%")
MyShortcut.WindowStyle = 4
MyShortcut.IconLocation = _
WSHShell.ExpandEnvironmentStrings("%windir%\calc.exe, 0")
MyShortcut.Save
```

Для запуска сценария RemoteShortcut.vbs на удаленном компьютере "\\Stand" нужно создать другой сценарий RunRemoteScript.vbs (рис 4.2).

В этом сценарии вначале создается объект WshController:

```
Set Controller = WScript.CreateObject("WshController").
```

Затем мы получаем ссылку на экземпляр объекта WshRemote на машине "\\Stand", соответствующий сценарию с текстом, взятым из файла D:\RemoteScript.js:

```
Set RemScript = Controller.CreateScript("D:\\RemoteScript.js", "stand")
Запускается удаленный сценарий с помощью метода Execute:
RemScript.Execute  ' Запустить удаленный сценарий
```

После этого нужно дождаться окончания работы сценария на удаленной машине, что делается путем контроля в цикле while свойства Status объекта WshRemote (значение свойства status, равное 2, говорит о том, что выполнение удаленного сценария завершено):

```
While RemScript.Status <> 2
  ' Цикл выполняется до завершения удаленного сценария
 WScript.Sleep 100  ' Приостанавитьм сценарий на 0,1 сек
```

Метод Sleep объекта WScript вызывается в цикле для того, чтобы освободить процессор во время ожидания завершения удаленного сценария.

```
'Имя файла - RemoteShortcut.vbs
'Автор - учебный
'Дата -20.03.2013
'Создать ярлык для калькулятора
'и поместить его в папку Автозагрузка
'*********************************************************************
' Создаем ярлык
  Dim WSHShell
 Set WSHShell = WScript.CreateObject("WScript.Shell")
  Dim MyShortcut, MyDesktop, StartupPath
  ' Узнаем путь к специальной папке Автозагрузка
 StartupPath = WSHShell.SpecialFolders("Startup")
  ' Создаем ярлык для Автозагрузки
 Set MyShortcut = WSHShell.CreateShortcut(StartupPath & _
 "\Ярлык к калькулятору.lnk")
  ' Устанавливаем свойства для ярлыка
 MyShortcut.TargetPath = WSHShell.ExpandEnvironmentStrings("%windir%\calc.exe")
 MyShortcut.WorkingDirectory = WSHShell.ExpandEnvironmentStrings("%windir%")
 MyShortcut.WindowStyle = 4
 MyShortcut.IconLocation = _
 WSHShell.ExpandEnvironmentStrings("%windir%\calc.exe, 0")
 MyShortcut.Save
```

Рис. 4.2. Листинг скрипта запуска удаленного сценария RunRemoteScript.vbs

Контролировать ход выполнения удаленных сценариев можно не только путем анализа свойства Status, но и с помощью обработки событий Start (запуск сценария), Error (ошибка при выполнении сценария) и End (завершение работы сценария) объекта WshRemote. Однако при контроле за ходом выполнения удаленного сценария с помощью обработки событий объекта WshRemote затрачивается больше ресурсов компьютера по сравнению с простой проверкой свойства Status. Кроме этого, при обработке событий увеличивается сетевой трафик между локальной и удаленной машинами.

Вопросы для самопроверки

1. Какие сценарии WSH называются удаленными?
2. Какой параметр системного реестра ОС разрешает исполнение удаленных сценариев?
3. Какой сервер WSH исполняет удаленный сценарий?
4. Какой объект WSH позволяет работать с удаленным компьютером?

5. БЕЗОПАСНОСТЬ ПРИ РАБОТЕ СО СЦЕНАРИЯМИ WSH

Одним из главных преимуществ WSH является возможность запуска программ-сценариев, которые хранятся в виде исходного текста, что максимально упрощает процессы написания и распространения программ – не нужны ни дополнительные компиляторы для создания исполняемого кода, ни специальные утилиты для установки и регистрации сценариев в операционной системе. Однако при использовании таких сценариев в силу той же простоты сразу возникает несколько проблем.

Во-первых, исходный код сценария является незащищенным – любой пользователь, запускающий сценарий, может модифицировать его и использовать в дальнейшем как свой собственный (нарушаются авторские права).

Во-вторых, простота распространения и выполнения сценариев открывает широкие возможности для написания вредоносных сценариев-вирусов, которые могут, например рассылаться по электронной почте, как широко известный вирус "*I Love You*".

Поэтому при использовании сценариев WSH вопросы безопасности имеют весьма большое значение. В этом разделе описано, каким образом можно создавать зашифрованные файлы сценариев и добавлять к сценариям цифровые подписи для указания автора сценария. Кроме этого, рассмотрен процесс организации политик безопасности для сценариев WSH, позволяющих, например, запретить выполнение неподписанных сценариев любого типа или вообще заблокировать WSH для определенных пользователей.

5.1. Шифрование сценариев

Начиная с версии 2.0, в WSH появилась возможность скрыть от пользователя исходный текст сценария, преобразовав (зашифровав) его с помощью программы Microsoft Script Encoder, которую можно свободно скачать по адресу: http://msdn.microsoft.com/scripting/vbscript/download/x86/sce10en.exe.

Программа Script Encoder может применяться для шифрования сценариев JScript (файлы *.js), VBScript (файлы *.vbs) и WS-файлов (расширение *.wsf), а также сценариев, содержащихся в гипертекстовых файлах HTML.

Шифрование с помощью Script Encoder не следует рассматривать как надежное средство сохранения в тайне исходного кода сценария – программа просто преобразует текст сценария в кодировку, непригодную для чтения, и профессионал сможет из него восстановить первоначальное содержимое. Однако для защиты сценария от изменений обычными пользователями подобного шифрования вполне достаточно.

Для запуска программы Script Encoder служит файл screnc.exe; по умолчанию установка исполняемого файла и файла помощи производится в каталог Program Files\Windows Script Encoder. Программа srcenc.exe

запускается из командной строки, в качестве ее обязательных параметров указываются имена исходного файла сценария и файла, в котором будет содержаться этот сценарий в зашифрованном виде.

В системе зарегистрированы специальные расширения для файлов с зашифрованными сценариями WSH: jse для сценариев JScript и vbe – для сценариев VBScript

```
'*********************************************
' Имя: ForEncode.vbs
' Язык: VBScript
' Описание: Исходный текст сценария
'*********************************************
WScript.Echo "Привет!"
```

Рис 5.1. Исходный листинг сценария ForEncode.vbs

```
#@~^xQEAAA==vCeMCeCeeCeCMeCeMeCeMMCeeCMeCeeCMMeCeCeMeMMCeMeCMeCeMMCeeMM
CeMeCMCeM@#@&EPИмя),sK.2 mGN□R-
8kPP,~P,PP,~~P,P,P~P~~,P~P,~P,P~~,PP~~,P~P,~,P~,@#@&BPЯзык=Pj$?1DrwDP~~,PP,~P,PP,~~P,P,P~P~~
,P~P,~P,P~~,PP~~,P~P,~,P~,P@#@&B,Описание),Исходный~текстPсценария,P~P,P~~,PP,~P,PP,~~P,P,P~P
~~,P@#@&EMeCeMMCeeMMCeMeCMCeMCeCeeCeCMeCeMeCeMMCeeCMeCeeCMMeCeCeMeMMCeMeCM
eC@#@& UmDr2DR3m4G,JПриветZr@#@&BeCeCMeCeMeCeM~,КонецPeCMeCeeCMMeCeCeMeMMCeMeC
MeCeMMCeeMMCeMeCMCeMCe@#@&PEAAAA==^#~@
```

Рис. 5.2. Зашифрованный сценарий Encoded.vbe

Как видно из примеров рис. 5.1 и 5.2, символы кириллицы остаются в зашифрованных сценариях без изменения.

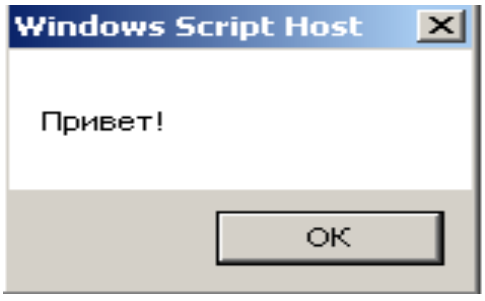


Рис. 5.3. Результаты выполнения зашифрованного сценария Encoded.vbe

Зашифрованные файлы Encoded.jse и Encoded.vbe можно запускать с помощью cscript.exe или wscript.exe, выполняться они будут точно так же, как и исходные сценарии (рис. 5.3).

Еще одной весьма полезной особенностью сценариев, зашифрованных с помощью Script Encoder, является то, что при запуске такого сценария автоматически производится контроль целостности файла. Например, если в файле Encoded.vbe убрать или добавить букву в слово «При-

вет», то при запуске будет выведено сообщение об ошибке и сценарий выполняться не будет.

Вопросы для самопроверки

1. Какое расширение имеют файлы с зашифрованными текстами сценариев на VBS?
2. Как в WSH сценариях шифруются символы кириллицы?
3. С помощью какой программы можно зашифровать сценарии WSH?
4. Напишите команду, которая шифрует сценарий A.vbs.

5.2. Параметры реестра и политика безопасности для WSH

Режим выполнения сценариев WSH зависит от нескольких параметров системного реестра, которые могут быть записаны в двух разделах:

HKLM\Software\Microsoft\Windows Script Host\Settings (1)

или

HKCU\Software\Microsoft\Windows Script Host\Settings (2)

В разделе (1) хранятся установки WSH для всех пользователей, запускающих сценарии на данной машине, а в разделе (2) – для текущего пользователя, зарегистрированного в системе. При этом строковый параметр IgnoreUserSettings из раздела (1) определяет, откуда именно будут браться параметры: если IgnoreUserSettings равен 0 или вообще не задан, то на политику безопасности для сценариев WSH будут влиять параметры из раздела (1). Если же IgnoreUserSettings равен 1, то параметры берутся из раздела (2).

Параметры реестра, определяющие политику безопасности при использовании сценариев WSH, описаны в табл. 5.1.

Таблица 5.1

Параметры реестра для WSH

Параметр	Тип	Описание
Enabled	Строковый (REG_SZ)	Если значение равно 0, то выполнение любых сценариев WSH запрещено. Если значение равно 1, то на машине могут выполняться локальные сценарии WSH. Значением по умолчанию является 1

Remote	Строковый (REG_SZ)	Если значение равно 0, то выполнение удаленных сценариев WSH запрещено. Если значение равно 1, то на машине могут выполняться удаленные (т. е. запускаемые с других компьютеров) сценарии WSH. Значением по умолчанию является 0

Окончание табл. 5.1

Параметр	Тип	Описание
TrustPolicy	Целый (REG_DWORD)	Если значение равно 0, то все сценарии запускаются без проверки их цифровой подписи. Если значение равно 1, то перед запуском неподписанных сценариев или сценариев с подписью, которой соответствует цифровой сертификат, не входящий в число доверяемых, будет выводиться диалоговое окно с предупреждением о возможной опасности такого сценария (при этом есть возможность отказаться от выполнения сценария). Если значение равно 2, то будут запускаться только сценарии, которые подписаны цифровой подписью, и к сертификату, с помощью которого создана эта подпись, установлено доверие. Значением по умолчанию является 0
UseWINSAFER	Строковый (REG_SZ)	В операционных системах Windows 9x/ME/NT/2000 этот параметр не используется. В Windows XP параметр определяет, следует ли к сценариям WSH применять политики ограниченного использования программ (SRP). Если значение равно 0, то к сценариям WSH политики ограниченного использования программ не применяются, а режим выполнения сценариев определяется параметром TrustPolicy. Если значение равно 1, то режим выполнения сценариев задается политикой ограниченного использования программ, а параметр TrustPolicy игнорируется. Значением по умолчанию является 0
LogSecurity Failures	Строковый (REG_SZ)	Если значение равно 1, то информация о всех ошибках, которые возникают при выполнении сценариев и связаны с вопросами безопасности, заносится в журнал событий системы (System Event Log). Другими словами, ведется аудит отказов для сценариев WSH. Если значение равно 0, то информация об ошибках в сценариях, которые связаны с нарушениями установленных политик безопасности, не заносится в журнал событий. Значением по умолчанию является 0
LogSecurity Successes	Строковый (REG_SZ)	Если значение равно 1, то ведется аудит успехов для сценариев WSH, т. е. в в журнал событий системы (System Event Log) записывается информация о каждом успешно запущенном сценарии. Если значение равно 0, то аудит успехов не ведется. Значением по умолчанию является 0

Устанавливая определенным образом значения параметров системного реестра из табл. 5.1, можно настроить политику безопасности при использовании сценариев WSH для конкретного пользователя или рабочей станции (например, можно разрешить выполнение сценариев только администраторам домена). В сетях Windows NT или на автономной машине для этого используется редактор системной политики (Poledit.exe), а в сетях с установленной службой каталогов Active Directory применяется групповая политика (Group Policy), доступная с помощью оснастки Active Directory – пользователи и компьютеры (Active Directory – users and computers) консоли управления MMC. На автономном компьютере с операционной системой Windows XP/2000 для настройки политики безопасности WSH можно также изменять локальную групповую политику, которая позволяет задать одинаковые параметры безопасности для всех пользователей данного компьютера.

Вопросы для самопроверки

1. Как запретить на компьюторе исполнение любых сценариев WSH?
2. Как запретить на компьюторе исполнение удаленнных сценариев WSH?
3. Каким образом обеспечить запись в журнал событий ОС информации о каждом успешно запущенном сценарии WSH?

5.3. Защита сценариев WSH с помощью цифровой подписи

Сценарии WSH можно защищать с помощью цифровой подписи, которая позволяет определить происхождение сценария и гарантировать его целостность, т. е. отсутствие в этом сценарии несанкционированных изменений. Если источник, из которого поступил этот сценарий, является надежным (trusted source), то сценарий можно выполнять. В случае же ненадежности создателя или распространителя сценария можно отказаться от запуска сценария.

Таким образом, разработчик должен идентифицировать свои сценарии цифровой подписью, чтобы пользователи, получающие такие сценарии, знали, где и кем они были разработаны. После добавления цифровой подписи к сценарию он может свободно запускаться (но не модифицироваться!) всеми, кто указал автора программы как надежного источника сценариев.

Цифровая подпись в Windows создается с помощью цифровых сертификатов. Рассмотрим, каким образом можно получить цифровой сертификат и установить доверие к этому сертификату.

Цифровой сертификат – это электронный документ, который однозначно идентифицирует его владельца.

Сертификаты различных типов широко используются во многих службах безопасности Windows для разных целей, в частности:

- для шифрования и защиты от искажений данных, которые передаются по локальной сети или с помощью электронной почты;
- для подписания электронных писем (получатель сообщения может проверить, что сообщение не было изменено в процессе доставки и что сообщение пришло именно от данного отправителя) и др.

Различаются цифровые сертификаты трех типов: созданные разработчиком, выданные разработчику организацией и полученные от центра сертификации.

Цифровой сертификат, созданный разработчиком, обычно используют те пользователи, которые доверяют этому разработчику. Например, администратор локальной сети может создать свой собственный сертификат для подписи сценариев WSH, которые он создает и распространяет внутри своей организации.

В организации может быть организован свой сервер выдачи цифровых сертификатов. Таким образом, организация распространяет сертификаты среди собственных разработчиков, не прибегая к услугам коммерческих центров сертификации.

Наиболее быстрым способом создания собственного цифрового сертификата является использование программы SelfCert.exe, входящей в состав Microsoft Office 2000/XP или программы «Цифровой сертификат для проектов VBA» из средств Microsoft Office 2007. Запустив эту утилиту, мы получим диалоговое окно, позволяющее задать имя создаваемого сертификата (рис. 5.4). В качестве этого имени можно использовать, например свое имя или название организации.

Результат работы утилиты представлен на рис. 5.5.

Для того чтобы оценить свойства созданного сертификата, необходимо запустить консоль управления Microsoft Management Console (MMC) – инструмент для создания, сохранения и открытия средств администрирования (называемых консолями (Snap-in) MMC).

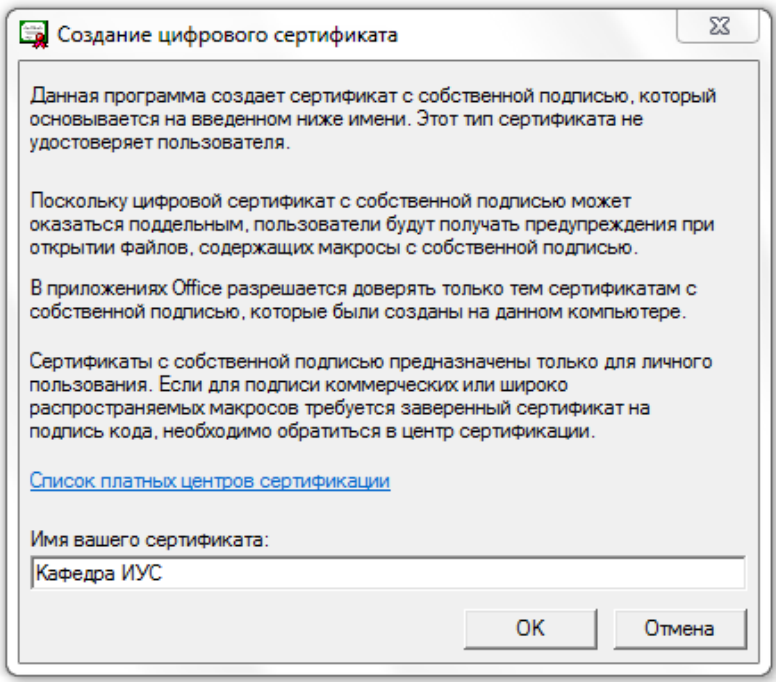


Рис. 5.4. Запуска программы «Цифровой сертификат для проектов VBA»

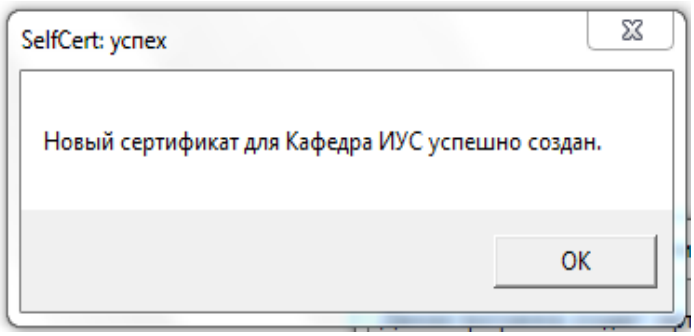


Рис. 5.5. Результаты работы программы
«Цифровой сертификат для проектов VBA»

Для запуска MMC нужно выполнить команду mmc либо в командной строке, либо с помощью пункта Выполнить (Run) меню Пуск (Start). В результате на экране появится окно новой консоли (рис. 5.6).

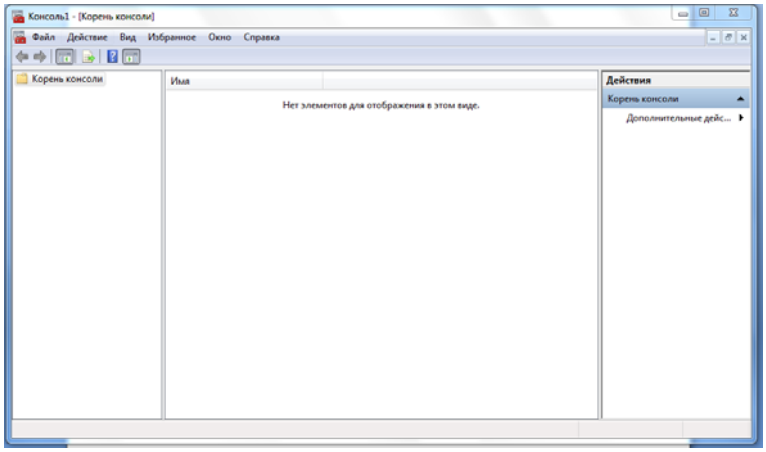


Рис. 5.6. Новая консоль MMC

Теперь следует добавить в консоль оснастку Сертификаты (Certificates). Для этого нужно в меню Консоль (Console) выбрать пункт Добавить/удалить оснастку (Add/Remove Snap-in), после чего появится диалоговое окно, показанное на рис. 5.7.

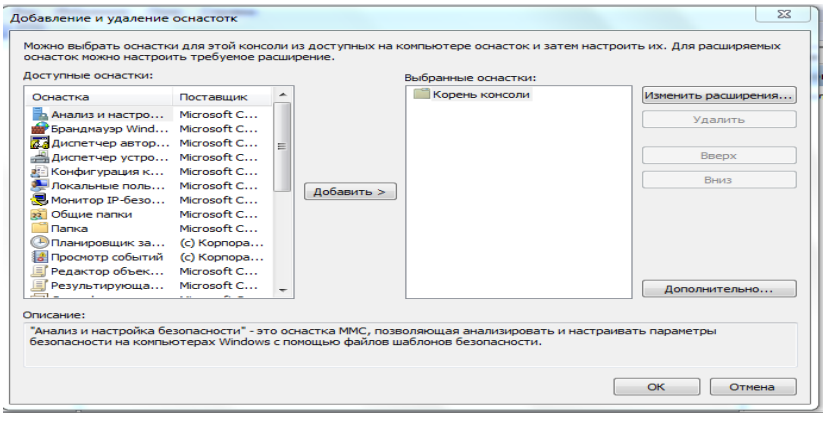


Рис. 5.7. Диалоговое окно
«Добавление и удаление оснасток»

Из списка «Доступные оснастки» нужно выбрать Сертификаты (Certificates) и нажать кнопку Добавить (Add). После этого в новом диалоговом окне (рис. 5.8) отметим, что добавляемая оснастка будет управлять сертификатами для своей учетной записи и далее следует нажать кнопку Готово (Finish).

Никаких других оснасток в окно консоли мы добавлять не будем, поэтому нажимаем кнопку Закрыть (Close) в списке оснасток и кнопку OK в окне Добавления/удаления оснасток.

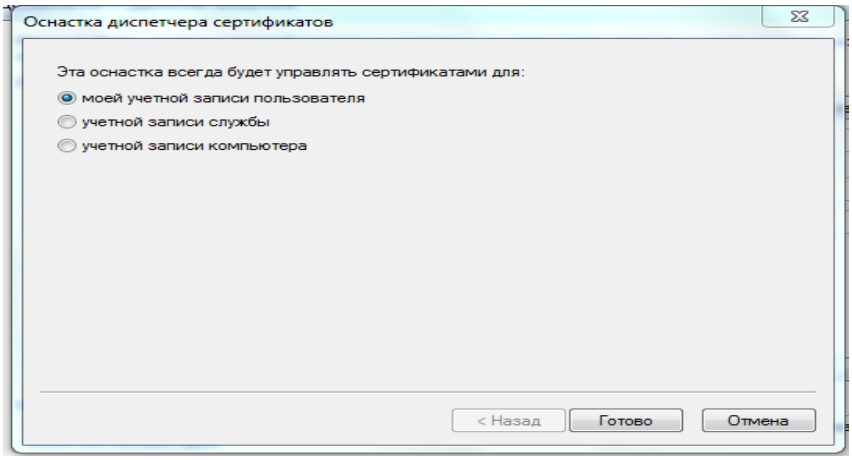


Рис. 5.8. Выбор типа сертификатов,
которым будет управлять добавляемая оснастка

Созданные сертификаты будут находиться в разделе Личные – Сертификаты (Personal – Sertificates) (рис. 5.9). Выделив нужный сертификат в списке и нажав клавишу <Enter>, мы выведем окно с описанием свойств сертификата (рис. 5.10).

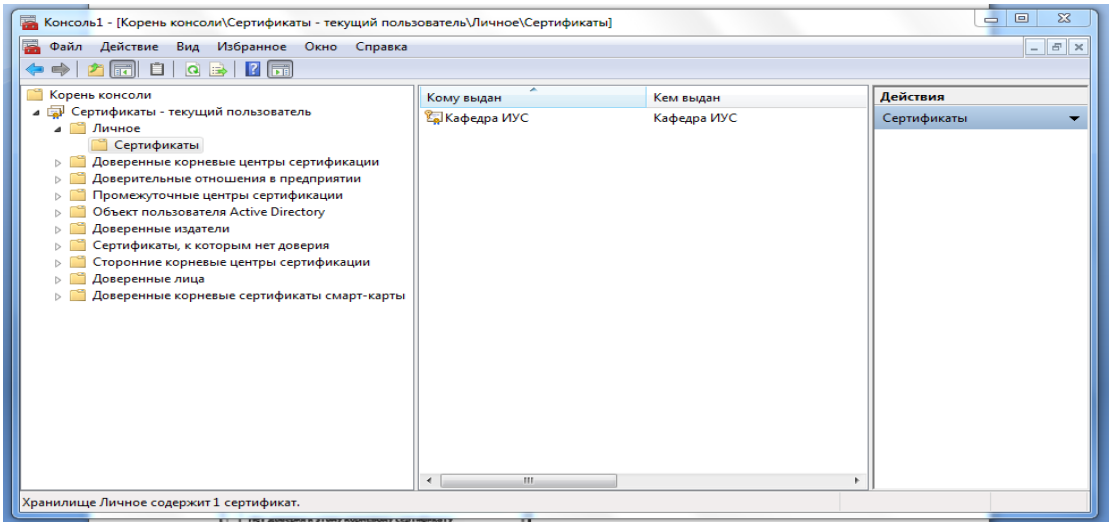


Рис. 5.9. Расположение личных сертификатов

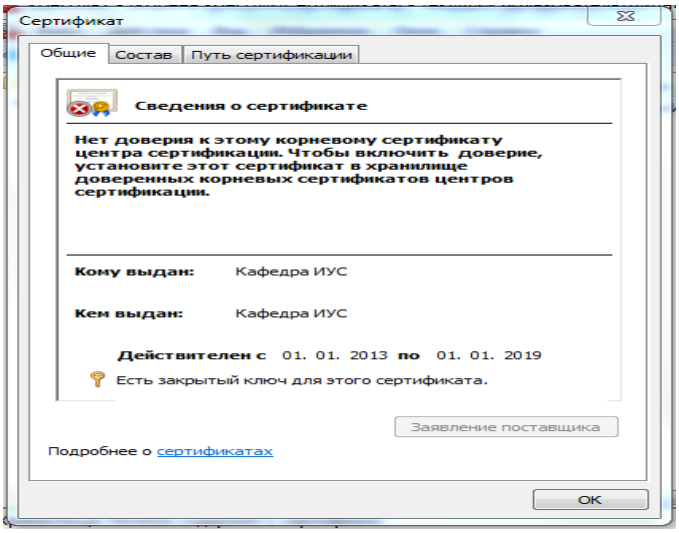


Рис. 5.10. Свойства сертификата

Как видно, для нового сертификата не установлено доверие. Для того чтобы установить доверие к сертификату («Кафедра ИУС»), достаточно просто перетащить с помощью мыши этот сертификат в раздел Доверенные корневые центры сертификации (рис. 5.11).

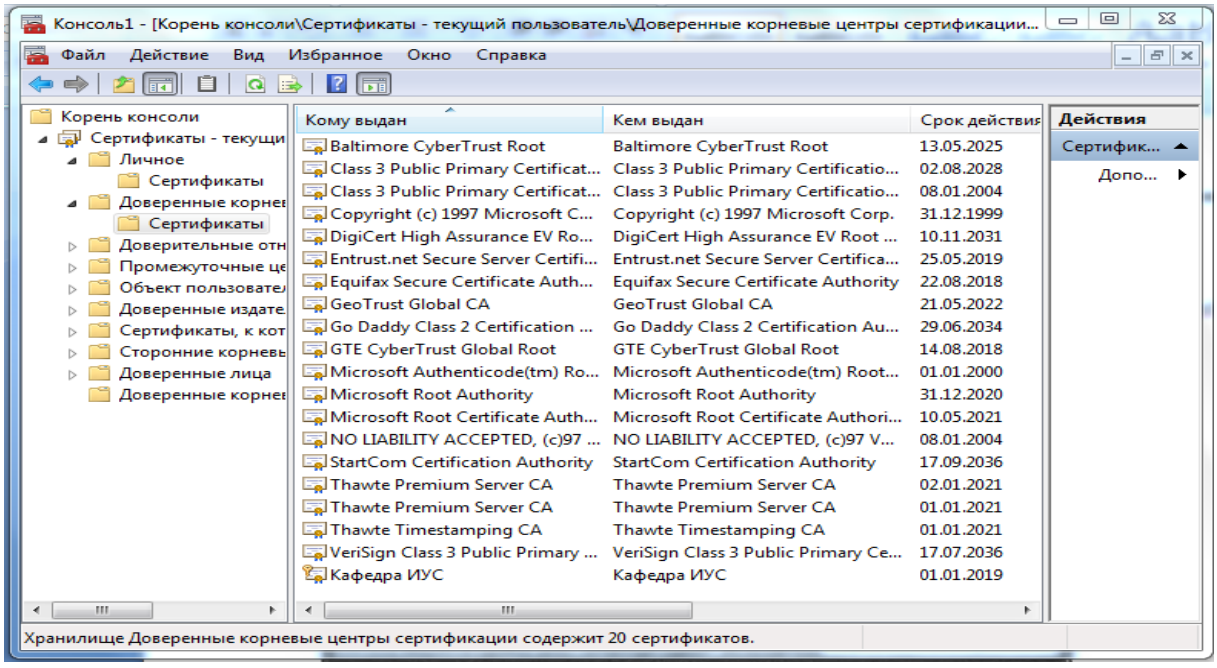


Рис. 5.11. Сертификаты, к которым установлено доверие

Свойства сертификата «Кафедра ИУС» после установки доверия к нему показаны на рис. 5.12.

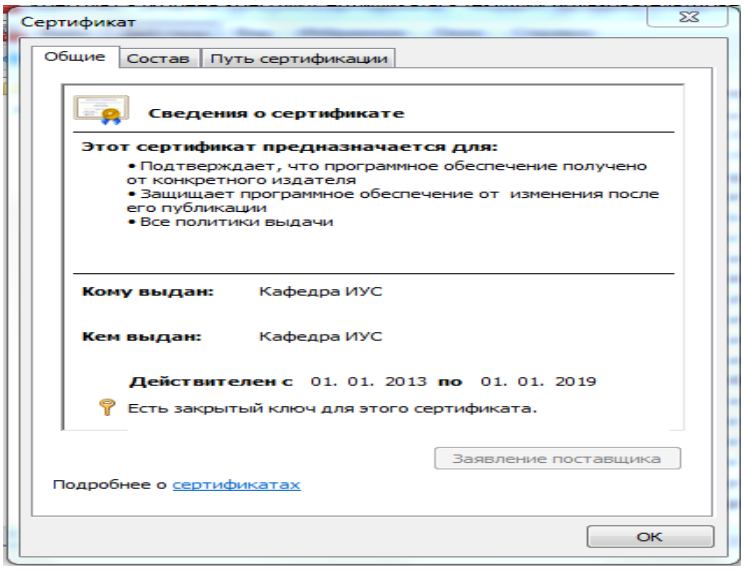


Рис. 5.12. Свойства сертификата,
к которому установлено доверие

Выполним теперь экспорт в файл сертификата «Кафедра ИУС». Для этого следует выделить нужный сертификат и выбрать в меню Действие | Все задачи (Action | All Tasks) пункт Экспорт (Export), после чего запустится мастер экспорта сертификатов, в котором нужно согласиться со всеми настройками, предлагаемыми по умолчанию, а в качестве имени экспортируемого файла указать "C:\Пользователи\ГУБИН\УЧ_ПОС_WSH \Sert\ IUS.cer".

ПРИЛОЖЕНИЕ

Справочник по языку VBScript

Язык VBScript (Visual Basic Script Edition) является урезанной версией языка Microsoft Visual Basic, поэтому для тех, кто программировал на Visual Basic или VBA, язык VBScript окажется очень знакомым.

Строки кода и комментарии

В отличие от JScript, для сценариев VBScript в конце строки не нужно ставить точку с запятой. В случае необходимости написания одного оператора на нескольких строках в конце этих строк нужно ставить символ подчеркивания "_":

```
s = "Символьная " & _
 "строка"
```

Комментарием в VBScript считаются все символы в строке, идущие после символа апострофа ' или ключевого слова Rem, например:

```
Rem Этот комментарий занимает всю строку
theSum=1 'А этот – часть строки
```

Переменные

Переменные в VBScript могут быть глобальными (доступными из любого места сценария) и локальными (область действия ограничивается блоком кода, в котором они определены). Все переменные VBScript имеют стандартный тип Variant. Объявляются переменные обычно с помощью ключевого слова Dim, например:

```
Dim MyVariable.
```

По умолчанию переменные в VBScript можно предварительно не объявлять; для включения режима обязательного объявления переменных нужно вставить в самую первую строку сценария выражение Option Explicit.

Язык VBScript является регистро-независимым, т. е. имена MyVariable и myvariable представляют одинаковые переменные. При выборе имен переменных следует придерживаться следующих правил: имя переменной должно начинаться с буквы и не должно содержать символа "," и превышать 255 символов.

Значения переменным в VBScript присваиваются с помощью оператора "=", например:

```
Dim MyVariable MyVariable = "Привет!".
```

Здесь мы объявили переменную MyVariable и записали в нее текстовую строку. Отметим, что далее в любом месте сценария мы можем присвоить переменной MyVariable, скажем, числовое значение, например:

```
MyVariable = 10.
```

Подтипы данных

Хотя в VBScript определен только один тип Variant, внутри этого типа имеется разделение на подтипы, описание которых приведено в табл. П.1. Для преобразования переменных к определенному подтипу нужно использовать соответствующую функцию преобразования; такие функции также представлены в табл. П.2.

В переменную, которая была ранее объявлена с использованием ключевого слова Dim, можно записать ссылку на какой-либо объект.

Таблица П1

Подтипы данных

Подтип	Функция преобразования	Описание
Empty	–	Автоматически присваивается новым переменным, когда для них еще не определено явное значение
Null	–	Указывает на то, что переменная не содержит допустимых значений
Bool	CBool(x)	Используется для работы с логическими переменными, принимающим два допустимых значения: true или false
Byte	CByte(x)	Содержит целые числа в диапазоне от 0 до 255
Integer	CInt(X)	Содержит целые числа в диапазоне от –32768 до 32768
Currency	CCur(x)	Специальный числовой формат для денежных величин
Long	CLng(x)	Содержит целые числа в диапазоне от –2147483648 до 2147483647
Single	CSngl(x)	Тип чисел с плавающей точкой одинарной точности
Double	CDbl(x)	Тип чисел с плавающей точкой двойной точности
Date/Time	CDate(x)	Содержит числа, соответствующие датам и времени от 1 января 100 года до 31 декабря 9999 года
String	CStr(x)	Символьный подтип данных. Текстовые строки в VBScript – это последовательность символов, заключенных в двойные кавычки
Object	–	Ссылка на объект
Error	–	Тип данных, предназначенный для хранения номеров ошибок

Делается это с помощью оператора Set, например:

```
Dim FSO
Set FSO=CreateObject("Scripting.FileSystemObject")
```

Здесь функция CreateObject() возвращает экземпляр объекта FileSystemObject, ссылка на который заносится в переменную FSO.

После того как ссылка на объект станет ненужной, переменную можно освободить с помощью ключевого слова Nothing:

```
Set FSO=Nothing
```

Константы

Пользовательские константы в VBScript объявляются с помощью ключевого слова Const, например:

```
Const MyConst="Это моя константа".
```

Кроме этого, VBScript поддерживает много встроенных именованных констант (их не нужно дополнительно объявлять в сценарии), применение которых упрощает использование различных внутренних функций (например, MsgBox() или InputBox()). Имена, значения и описания внутренних констант приведены в табл. П.2 – П.9.

Таблица П.2

Константы для обозначения цветов

Имя	Значение	Описание
vbBlack	&h00	Черный цвет
vbRed	&hFF	Красный цвет
vbGreen	&hFF00	Зеленый цвет
vbYellow	&hFFF	Желтый цвет
vbBlue	&hFF0000	Синий цвет
vbMagenta	&hFF00FF	Фиолетовый цвет
vbCyan	&hFFFF00	Бирюзовый цвет
vbWhite	&hFFFFFF	Белый цвет

Таблица П.3

Константы для нумерации дней недели

Имя	Значение	Описание
vbSunday	1	Воскресенье
vbMonday	2	Понедельник
vbTuesday	3	Вторник
vbWednesday	4	Среда
vbThursday	5	Четверг
vbFriday	6	Пятница
vbSaturday	7	Суббота

Таблица П.4

Константы для определения первого дня в неделе и первой недели в году

Имя	Значение	Описание
vbUseSystemDayOfWeek	0	Использовать для определения первого дня недели региональные настройки системы
vbFirstJan1	1	Первой неделей в году считается та, в которой было 1 января
vbFirstFourDays	2	Первой неделей в году считается та, в которой было по крайней мере четыре дня нового года
vbFirstFullWeek	3	Первой неделей в году считается первая полная неделя

Таблица П.5

Константы для работы с датой и временем

Имя	Значение	Описание
vbGeneralDate	0	Дата и время выводятся в формате, определяемом региональными настройками системы
vbLongDate	1	Выводить дату, используя полный формат
vbShortDate	2	Выводить дату, используя краткий формат
vbLongTime	3	Выводить время, используя полный формат
vbShortTime	4	Выводить время, используя краткий формат

Таблица П.6

Константы для диалоговых окон

Имя	Значение	Описание
vbOkOnly	0	Выводится кнопка OK
vbOkCancel	1	Выводятся кнопки OK и Отмена (Cancel)
vbAbortRetryIgnore	2	Выводятся кнопки Стоп (Abort), Повтор (Retry) и Пропустить (Ignore)
vbYesNoCancel	3	Выводятся кнопки Да (Yes), Нет (No) и Отмена (Cancel)
vbYesNo	4	Выводятся кнопки Да (Yes) и Нет (No)
vbRetryCancel	5	Выводятся кнопки Повтор (Retry) и Отмена (Cancel)
vbCritical	16	Выводится значок Stop Mark
vbQuestion	32	Выводится значок Question Mark
vbExclamation	48	Выводится значок Exclamation Mark
vbInformation	64	Выводится значок Information Mark
vbDefaultButton1	0	По умолчанию в окне выбирается первая кнопка
vbDefaultButton2	256	По умолчанию в окне выбирается вторая кнопка
vbDefaultButton3	512	По умолчанию в окне выбирается третья кнопка
vbDefaultButton4	768	По умолчанию в окне выбирается четвертая кнопка
vbApplicationModal	0	Диалоговое окно выводится в модальном режиме

Таблица П.7

Результаты нажатия кнопок в диалоговых окнах

Имя	Значение	Описание
vbOk	1	Нажата кнопка OK
vbCancel	2	Нажата кнопка Отмена (Cancel)
vbAbort	3	Нажата кнопка Стоп (Abort)
vbRetry	4	Нажата кнопка Повтор (Retry)
vbIgnore	5	Нажата кнопка Пропустить (Ignore)
vbYes	6	Нажата кнопка Да (Yes)
vbNo	7	Нажата кнопка Нет (No)

Таблица П.8

Константы для обозначения подтипов данных

Имя	Значение	Описание
vbEmpty	0	Переменная не инициализирована
vbNull	1	Переменная не содержит корректных данных
vbInteger	2	Переменная имеет подтип Integer
vbLong	3	Переменная имеет подтип Long
vbSingle	4	Переменная имеет подтип Single
vbDouble	5	Переменная имеет подтип Double
vbCurrency	6	Переменная имеет подтип Currency
vbDate	7	Переменная имеет подтип Date
vbString	8	Переменная имеет подтип String
vbObject	9	Переменная имеет подтип Object
vbError	10	Переменная имеет подтип Error
vbBoolean	11	Переменная имеет подтип Boolean
vbVariant	12	Переменная имеет подтип Variant (только для массивов переменных типа Variant)
vbDataObject	13	Объект доступа к данным
vbDecimal	14	Переменная имеет подтип Decimal
vbByte	17	Переменная имеет подтип Byte
vbArray	8192	Переменная является массивом

Таблица П.9

Прочие константы

Имя	Значение	Описание
vbCr	Chr(13)	Возврат каретки
vbCrLf	Chr(13) & Chr(10)	Возврат каретки и перевод строки
vbFormFeed	Chr(12)	Перевод страницы
vbLf	Chr(10)	Перевод строки
vbNullChar	Chr(0)	Символ с нулевым кодом
vbNullString	Нулевая строка	Нулевая строка
vbTab	Chr(9)	Символ табуляции
vbVerticalTab	Chr(11)	Символ вертикальной табуляции

Окончание табл. П 9

Имя	Значение	Описание
vbUseDefault	–2	Использовать значения по умолчанию из региональных настроек системы
vbTrue	–1	Логическое значение «истина»
vbFalse	0	Логическое значение «ложь»
vbObjectError	–2147221504	Определяет минимальное значение для номеров ошибок, задаваемых пользователем

Массивы

Массивы в VBScript могут быть двух видов: статические (фиксированной длины) и динамические (переменной длины). Объявляются массивы, как и обычные переменные, с помощью ключевого слова Dim.

Для объявления статического массива нужно после его названия указать в круглых скобках наибольшее значение, которое может принимать индекс элемента в этом массиве, например:

```
Dim MyArr(10).
```

В языке VBScript (в отличие, например, от VBA) нумерация в массивах всегда начинается с нуля, поэтому объявленный выше массив MyArr будет содержать 11 элементов, обращаться к которым нужно следующим образом:

```
MyArr(0)="Это первый элемент"
MyArr(1)="Это второй элемент"
MyVar=MyArr(0).
```

Можно объявить двумерный массив, указав максимальные значения индексов для строк и столбцов соответственно, например:

```
Dim MyArr(5, 10) 'Массив из 6 строк и одиннадцати столбцов
```

При объявлении динамического массива его размеры в круглых скобках не указываются:

```
Dim MyArr()
```

Для использования динамического массива в сценарии применяется оператор ReDim, который определяет конкретную длину массива, например:

```
ReDim MyArray(10)
```

После этого к элементам динамического массива можно обращаться так же, как и к элементам обычного:

```
MyArr(0)="Это первый элемент"
MyArr(1)="Это второй элемент"
```

Отметим, что размеры динамического массива можно менять неоднократно (с помощью того же ReDim). Для сохранения при этом содержимого массива следует в операторе ReDim использовать ключевое слово Preserve, например:

```
ReDim Preserve MyArray(20)
```

Операторы

В VBScript поддерживаются операторы, описание которых приведено ниже.

Арифметические операторы

Арифметические операторы языка VBScript представлены в табл. П.10.

Таблица П.10

Арифметические операторы

Оператор	Описание
- (унарный	Изменение знака аргумента на противоположный
- (бинарный	Вычитание двух чисел
+	Сложение двух чисел
*	Умножение двух чисел
/	Деление двух чисел
\	Целочисленное деление двух чисел
Mod	Вычисление остатка от деления двух чисел
^	Оператор возведения в степень

Операторы отношения и логические операторы

Операторы отношения используются для сравнения значений двух переменных. Эти операторы, описанные в табл. П.11, могут возвращать только логические значения true или false.

Таблица П.11

Операторы отношения

Оператор	Условие, при котором возвращается true
>	Левый операнд больше правого
>=	Левый операнд больше или равен правому
<	Левый операнд меньше правого
<=	Левый операнд меньше или равен правому
=	Левый операнд равен правому
<>	Левый операнд не равен правому

Внутри условных операторов могут применяться логические операторы (табл. П.12).

Таблица П.12

Логические операторы

Оператор	Описание
Not	Оператор отрицания. Возвращает true, если операнд равен false. В противном случае возвращает false
Or	Оператор отношения "ИЛИ". Возвращает true, если один из операндов равен true. В противном случае возвращает false
Xor	Оператор отношения "ИСКЛЮЧАЮЩЕЕ ИЛИ". Возвращает true, если один из операндов равен true, а другой равен false. В противном случае возвращает false
And	Оператор отношения "И". Возвращает true, если оба операнда равны true. В противном случае возвращает false

Условные операторы

В VBScript поддерживается условный оператор If…Then…Else. Общий вид этого оператора:

```
If условие_1 Then
 выражение_1
[ElseIf условие_2 Then
 выражение_2]
…
[Else
 выражение_3]
End If
```

При выполнении оператора If…Then…Else оценивается логическое условие (условие_1), стоящее после ключевого слова If. Если в результате оценки условия получилось значение true, то выполняется выражение_1 и происходит выход из оператора. В противном случае начинают по очереди проверяться условия, стоящие после ключевых слов ElseIf; если одно из этих условий истинно, то выполняется соответствующее выражение, после чего управление передается следующему после End If оператору. Если ни одно из проверяемых условий не является истинным, выполняется выражение, стоящее после ключевого слова Else, например:

```
If (theCount > 1) Then
 theMoments = "До взрыва осталось " & theCount & " сек!"
Else
 theMoments = "Осталась секунда!"
End If
```

Другим оператором, позволяющим производить выбор из нескольких вариантов, является Select Case. Синтаксис этого оператора:

```
Select Case выражение
Case значение_1
 выражение
[Case значение_2
 выражение_2]
...
[Case Else
 выражение_3]
End Select
```

Здесь сначала вычисляется значение выражения, которое затем по очереди сравнивается со значениями, стоящими после ключевых слов Case. В случае совпадения выполняются операторы в соответствующем блоке Case. Если ни одно из имеющихся значений не совпадает со значением выражения, то выполняются операторы, стоящие после слова Case Else.

Пример использования оператора Select Case:

```
Select Case MyVar
Case vbRed
 Color = "Красный"
Case vbGreen
 Color = "Зеленый"
Case vbBlue
 Color = "Синий"
Case Else
 Color = "Цвет непонятен"
End Select
```

Операторы циклов

В VBScript поддерживаются несколько типов циклов: цикл For…Next, цикл Do…Loop, цикл While…Wend, цикл For Each…Next.

Цикл For…Next

В общем случае оператор цикла For…Next записывается следующим образом:

```
For counter=start To end [Step step]
 тело цикла
 [Exit For]
 тело цикла
Next
```

Параметр counter здесь является счетчиком цикла; start – начальное значение этого счетчика; end – конечное значение; step – шаг приращения счетчика. Если ключевое слово Step не указано, то шаг приращения берется равным единице. Выход из цикла For…Next происходит, когда значение счетчика counter становится больше, чем значение параметра end. Выражение Exit For используется для безусловного выхода из цикла.

Ниже приведен пример использования цикла for…next:

```
Dim howFar 'Верхний предел для счетчика цикла
Dim sum(10) 'Массив из 11 элементов, индексы от 0 до 10
Dim icount, theSum
howFar = 10
theSum = 0
sum(0) = 0
'Цикл выполнится 11 раз
For icount = 0 To howFar
 theSum=theSum+icount
 sum (icount) = theSum
Next
```

Цикл For Each…Next

Оператор цикла For Each…Next предназначен для перебора всех элементов массива или коллекции:

```
For Each element In group
тело цикла
 [Exit For]
 тело цикла
Next [element]
```

Здесь параметр element является переменной, в которую будет записываться значение текущего элемента массива или коллекции при итерациях; group — имя массива или коллекции объектов.

Замечание. Напомним, что в JScript для перебора всех элементов коллекции необходимо использовать вспомогательный объект Enumerator.

С помощью оператора Exit For можно осуществить немедленный выход из цикла.

Ниже приведен пример использования цикла For Each…Next:

```
'Объявляем переменные
Dim FSO, Folder, Files, File, s
s = "Список файлов" & vbCrLf
'Создаем объект FileSystemObject
Set FSO = CreateObject("Scripting.FileSystemObject")
```

```
' Создаем объект Folder для корневого каталога диска C:
Set Folder = FSO.GetFolder("C:\")
'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
'Выводим сформированную строку на экран
WScript.Echo s
```

Цикл While…Wend

Цикл While…Wend записывается в следующем виде:

```
While условие
 тело цикла
Wend
```

Таким образом, в цикле While…Wend условие выполнения тела цикла проверяется перед началом очередной итерации. Если условие равно true, то тело цикла выполняется, в противном случае цикл завершается.

Пример использования цикла While…Wend приведен ниже:

```
Dim theMoments, theCount
theMoments = ""
theCount = 42 'Начальное значение счетчика цикла
While (theCount >= 1)
 If (theCount > 1) Then
  theMoments = "До взрыва осталось " & theCount & " сек!"
 Else
  theMoments = "Осталась секунда!"
 End If
 theCount = theCount - 1 'Уменьшаем значение счетчика
Wend
theMoments = "ВЗРЫВ!"
```

Цикл Do…Loop

Этот цикл может применяться в двух видах (с предусловием, которое проверяется до начала очередной итерации, и с пост-условием, которое проверяется после окончания итерации):

```
Do [While | Until] условие
 тело цикла
```

```
 [Exit Do]
 тело цикла
Loop
```

или

```
Do
 тело цикла
 [Exit Do]
 тело цикла
Loop [While | Until] условие
```

Если в цикле используется ключевое слово While, то итерации продолжаются до тех пор, пока условие равно true; если применяется Until, то как только значением условия станет true, произойдет выход из цикла.

Оператор Exit Do позволяет выйти из цикла до завершения его итераций. Пример использования цикла Do…Loop приведен ниже:

```
Dim howFar 'Верхний предел для счетчика цикла
Dim sum(10) 'Массив из 11 элементов, индексы от 0 до 10
Dim icount, theSum
howFar = 10
theSum = 0
sum(0) = 0
'Цикл выполнится 11 раз
Do
 theSum = theSum+icount
 sum(icount) = theSum
 icount=icount+I
Loop While (icount < howFar)
```

Прочие операторы

В табл. П.13 приведены еще несколько достаточно часто применяемых операторов.

Таблица П.13

Прочие операторы

Оператор	Описание
.	Точка. Применяется для доступа к свойству объекта или для вызова его метода
( )	Скобки. Применяются либо для изменения порядка вычисления выражений, либо для передачи параметров функциям, либо для индексирования массива
&	Оператор конкатенации (склеивание между собой) символьных строк
With…End With	Позволяет обращаться к свойствам объекта без написания имени этого объекта

Обработка исключительных ситуаций

Режим обработки исключительных ситуаций в VBScript включается с помощью оператора On Error Resume Next. Если после этого при исполнении какою-либо оператора в сценарии произойдет ошибка времени выполнения, то управление передастся к следующему оператору в тексте.

Для анализа ошибок используется специальный объект Err, который содержит два свойства: Number – числовой код возникшей ошибки и Description – краткое описание этой ошибки.

В качестве примера приведем часть сценария, в которой происходит обработка исключительных ситуаций при подключении сетевого диска:

```
On Error Resume Next ' Включаем обработку ошибок времени выполнения
' Подключаем сетевой диск
WshNetwork.MapNetworkDrive Drive, NetPath
If Err.Numbero<>0 Then
 Mess="Ошибка при подключении диска " & Drive & " к " & NetPath &_
  "Код ошибки: " & e.number & "Описание: " & e.description
 WshShell.Popup Mess, 0, "Подключение сетевого диска", vbCritical
Else
 ' Все в порядке
 Mess = "Диск " & Drive & " успешно подключен к " & NetPath
 WshShell.Popup Mess, 0, "Подключение сетевого диска", vbInformation
End If
```

Для отмены режима обработки исключительных ситуаций нужно выполнить оператор On Error Goto 0.

Процедуры и функции

VBScript поддерживаются два вида подпрограмм: встроенные функции и функции или процедуры пользователя.

Математические функции

Поддерживаемые VBScript математические функции приведены в табл. П.14.

Таблица П.14

Математические функции

Функция	Описание
Abs(*x*)	Возвращает абсолютное значение числа *x*
Atn(*x*)	Возвращает арктангенс числа *x*
Cos(*x*)	Возвращает косинус числа *x*
Exp(*x*)	Экспоненциальная функция, возвращает число e, возведенное в степень *x*
Int(*x*)	Возвращает целую часть числа *x*
Log(*x*)	Возвращает натуральный логарифм числа *x*

Окончание табл. П.14

Функция	Описание
Rnd[(*x*)]	Возвращает случайное число от 0 до 1
Round(x[, n])	Округляет число *x* с точностью до n знаков после запятой
Sgn(*x*)	Знаковая функция числа *x*
Sin(*x*)	Возвращает синус числа *x*
Sqr(*x*)	Вычисляет квадратный корень из числа *x*
Abs(*x*)	Возвращает абсолютное значение числа *x*
Atn(*x*)	Возвращает арктангенс числа *x*

Символьные функции

Наиболее часто используемые функции, с помощью которых можно производить различные операции над символьными строками, приведены в табл. П15.

Таблица П.15

Символьные функции

Функция	Описание
Asc(str)	Возвращает ASCII-код первого символа в строке str
Chr(code)	Возвращает символ с ASCII-кодом code
InStr([start,] str1, str2[, compare])	Возвращает индекс символа, с которого начинается первое вхождение подстроки str2 в строку str1. Параметр start задает номер символа, с которого следует начинать поиск. Если этот параметр не задан, то поиск производится с начала строки. Поиск производится слева направо. Параметр compare задает режим сравнения при обработке строк (0 – двоичное сравнение, 1 – текстовое сравнение)
InStrRev(str1, str2[, start[, compare]])	То же самое, что функция InStr, но поиск производится справа налево, т. е. возвращается номер последнего символа, с которого начинается вхождение подстроки str2 в строку str1
Join(list[,delim])	Возвращает строку, полученную в результате конкатенации подстрок, содержащихся в массиве list. Параметр delim задает символ, разделяющий подстроки (по умолчанию таким символом является пробел)
LCase(str)	Возвращает строку, в которой все алфавитные символы преобразованы к нижнему регистру
Left(str, len)	Возвращает len символов с начала строки str
Len(str)	Возвращает число символов в строке str
LTrim(str), RTrim(str), Trim(str)	Удаляет из строки str начальные, конечные или и те и другие пробелы соответственно
Mid(str, start[, len])	Возвращает из строки str подстроку, которая начинается с позиции start и имеет длину len. Если параметр len не указан, то возвращаются все символы, начиная с позиции start до конца строки str

Окончание табл. П.15

Функция	Описание
Replace(expr, find, replacewith[, start[, count[, compare]]])	Возвращает строку, которая получается из строки expr путем замен входящих в нее подстрок find на подстроки replacewith. Параметр count определяет число подстрок, которые будут обработаны таким образом (по умолчанию производятся все возможные замены). Параметр compare задает режим сравнения при работе со строками (0 – двоичное сравнение, 1 – текстовое сравнение)
Right(str, len)	Возвращает len символов с конца строки str
Space(x)	Возвращает строку, состоящую из x пробелов
Split(Expr[, delim[, count[, compare]]])	Возвращает массив строк, полученных в результате разбиения строки Expr на подстроки. Параметр delim задает символ, разделяющий подстроки (по умолчанию таким символом является пробел). Параметр count определяет число подстрок, которые будут обработаны таким образом (по умолчанию в массив записываются все подстроки). Параметр compare задает режим сравнения при работе со строками (0 – двоичное сравнение, 1 – текстовое сравнение)
StrComp(str1, str2[, compare])	Возвращает число – результат сравнения строк str1 и str2. Если str1<str2, то возвращается –1; если str1=str2, то возвращается 0; если str1>str2, то возвращается 1. Параметр compare задает режим сравнения при работе со строками (0 – двоичное сравнение, 1 – текстовое сравнение)
String(number, char)	Возвращает строку, состоящую из number символов char
UCase(str)	Возвращает строку, в которой все алфавитные символы преобразованы к верхнему регистру

Для работы с датой и временем в VBScript имеется большой набор функций, основные из которых приведены в табл. П16.

Таблица П.16

Функции для работы с датой и временем

Функция	Описание
Date	Возвращает текущую системную дату
DateAdd(interval, number, date)	Возвращает дату, отстоящую от даты date на number интервалов, заданных параметром interval, который может принимать следующие значения: "yyyy" – год, "q" – квартал, "m" – месяц, "y" – день года, "d" – день, "w" – неделя, "ww" – неделя года, "h" – час, "m" – минута, "s" – секунда

Продолжение табл. П.16

Функция	Описание
DateDiff(interval, date1, date2[, firstdayofweek [, firstweekofyear]])	Возвращает разницу в интервалах interval (возможные значения этого параметра те же, что и в функции DateAdd) между датами date1 и date2. Параметр firstdayofweek – это константа, показывающая, какой из дней недели следует считать первым. Параметр firstweekofyear – это константа, показывающая, какую неделю следует считать первой в году
DatePart(interval, date [, firstdayofweek [, firstweekofyear]])	Возвращает ту часть даты date, которая соответствует параметру interval. Значения параметров interval, firstdayofweek и firstweekofyear здесь те же, что и в функции DateDiff
DateSerial(year, month, day)	Возвращает переменную подтипа Date, которая соответствует указанным году (параметр year), месяцу (параметр month) и дню (параметр day)
DateValue(date)	Возвращает переменную Variant подтипа Date, которая соответствует дате, заданной символьным параметром date
Hour(time)	Выделяет номер часа из даты или момента времени, заданных параметром time. Возвращает целое число от 0 до 23
IsDate(expr)	Возвращает true, если параметр expr задает корректную дату, и false в противном случае
Minute(time)	Выделяет количество минут из даты или момента времени, заданных параметром time. Возвращает целое число от 0 до 59
Month(date)	Выделяет номер месяца из даты, заданной параметром date. Возвращает целое число от 1 до 12
MonthName(month[, abbr])	Возвращает наименование для месяца с номером month. Если логический параметр abbr равен true, то наименование месяца представляется в виде аббревиатуры, в противном случае – в полном виде
Now	Возвращает текущие дату и время в виде, соответствующем региональным настройкам Windows
Time	Возвращает текущее системное время
Timer	Возвращает количество секунд, прошедших с полуночи
TimeSerial(hour, minute, second)	Возвращает переменную подтипа Date, которая соответствует указанным часу (параметр hour), минуте (параметр minute) и секунде (параметр second)
TimeValue(time)	Возвращает переменную подтипа Date, которая соответствует времени, заданному символьным параметром time

Окончание табл. П.16

Функция	Описание
Weekday(date[, firstdayofweek])	Возвращает целое число – день недели для даты, заданной параметром date. Параметр firstdayofweek – это константа, показывающая, какой из дней недели следует считать первым
WeekdayName(weekday[,abbr[, firstdayofweek]])	Возвращает наименование для дня недели с порядковым номером weekday. Если логический параметр abbr равен true, то наименование дня недели представляется в виде аббревиатуры, в противном случае – в полном виде. Значение параметра firstdayofweek здесь то же, что и в функции Weekday
Year(date)	Выделяет год из даты, заданной параметром date, и возвращает это целое число

Функции для работы с массивами

В табл. П.17 приведены функции, с помощью которых можно создавать новые массивы и получать сведения об уже имеющихся.

Таблица П.17

Функции для работы с массивами

Функция	Описание
Array(arglist)	Возвращает значение типа Variant, которое является массивом, составленным из элементов списка arglist. Отдельные элементы в arglist должны быть отделены друг от друга запятой
IsArray(varname)	Возвращает true, если переменная varname является массивом, и false в противном случае
LBound(arrayname[, dimension])	Возвращает наименьшее значение, которое может принимать индекс в массиве arrayname. Параметр dimension определяет, для какой именно размерности массива мы ищем это наименьшее значение (1 для первой размерности, 2 для второй размерности и т. д.). По умолчанию dimension равно 1
UBound(arrayname[, dimension])	Возвращает наибольшее значение, которое может принимать индекс в массиве arrayname. Параметр dimension определяет, для какой именно размерности массива мы ищем это наибольшее значение (1 для первой размерности, 2 для второй размерности и т. д.). По умолчанию dimension равно 1

Функции для работы с подтипами данных

При рассмотрении подтипов данных мы уже описывали функции конвертации, которые применяются для преобразования переменной к тому или иному подтипу (см. табл. П.9).

В табл. П.18 приведены функции, с помощью которых можно узнать, к какому подтипу принадлежит заданная переменная.

Таблица П.18

Функции для работы с подтипами данных

Функция	Описание
IsArray(expr)	Возвращает true, если параметр expr является массивом, и false в противном случае
IsDate(expr)	Возвращает true, если параметр expr задает корректную дату и false в противном случае
IsEmpty(expr)	Возвращает true, если переменная expr объявлена, но не инициализирована
IsNull(expr)	Возвращает true, если переменная expr не содержит никаких корректных данных
IsNumeric(expr)	Возвращает true, если выражение expr может быть рассмотрено в качестве числа, и false в противном случае
IsObject(expr)	Возвращает true, если переменная expr является указателем на внешний объект, и false в противном случае
VarType(varname)	Возвращает числовое значение, соответствующее подтипу переменной varname (см. табл. П2.8)

Прочие функции

Далее приведены еще несколько наиболее часто используемых функций (табл. П.19).

Таблица П.19

Некоторые прочие функции

Функция	Описание
CreateObject(servername.typename[, location])	Создает экземпляр объекта-сервера автоматизации и возвращает ссылку на него. Здесь servername – имя приложения, являющегося сервером; typename – тип или класс создаваемого объекта; location – сетевое имя компьютера, на котором будет создан объект
GetObject([pathname][, classname])	Возвращает ссылку на объект класса classname, который хранится в отдельном файле, путь к которому задается параметром pathname
Hex(number)	Возвращает шестнадцатеричное представление (в символьном виде) числа number

Окончание табл. П.19

Функция	Описание
InputBox(prompt[, title] [, default] [, xpos][,ypos] [, helpfile, context])	Выводит на экран диалоговое окно со строкой ввода и кнопками OK, Отмена и возвращает введенную в этом окне символьную строку. Параметр prompt задает сообщение, которое печатается перед строкой ввода; title определяет заголовок диалогового окна; default – значение, которое выводится по умолчанию в строку ввода. Параметры xpos и ypos определяют координаты левого верхнего угла окна. В случае необходимости элементам диалогового окна можно сопоставить контекстно-зависимую помощь. Параметр helpfile задает путь к файлу помощи, а число context – идентификатор содержания помощи
MsgBox(prompt[, buttons] [, title] [, helpfile, context])	Выводит на экран диалоговое окно с сообщением и различными кнопками и возвращает результат нажатия на одну из кнопок (возможные варианты возвращаемых функцией значений приведены в табл. П2.6). Параметр prompt задает сообщение, title определяет заголовок диалогового окна. Числовой параметр buttons определяет, какие именно кнопки должны быть представлены в окне (возможные значения этого параметра приведены в табл. П2.5). Параметры helpfile и context имеют то же значение, что и в функции InputBox
Oct(number)	Возвращает восьмеричное представление (в символьном виде) числа number

Функции и процедуры пользователя

Для определения процедуры, т. е. подпрограммы, которая не возвращает никакого значения, в VBScript используется конструкция Sub…End Sub. После названия процедуры в круглых скобках указывается список ее параметров, например:

```
Sub MyProcedure(Param1, Param2)
 Dim Sum
 Sum = Param1+Param2
End Sub
```

Если процедура не имеет параметров, то в скобках после имени ничего указывать не нужно:

```
Sub MyProcedure()
 …
End Sub
```

Вызывать процедуру из сценария можно двумя способами. Во-первых, можно просто написать имя нужной процедуры и указать через пробел список передаваемых параметров, например:

```
MyProcedure 3,10
```

Во-вторых, можно использовать специальный оператор Call, при этом список параметров обязательно должен быть заключен в круглые скобки:

```
Call MyProcedure(3, 10)
```

Для определения функции, т. е. подпрограммы, которая возвращает определенное значение, применяется конструкция Function…End Function. Как и при описании процедур, после названия функции в круглых скобках указывается список ее параметров, например:

```
Function MyFunction(Param1, Param2)
 …
End Function
```

Для того чтобы возвратить из функции какое-либо значение, нужно внутри функции присвоить это значение переменной, название которой совпадает с именем функции:

```
Function MyFunction(Param1, Param2)
Dim Sum
 Sum = Param1 + Param2
MyFunction = Sum
End Function
```

Если возвращаемое функцией значение не нужно присваивать никакой переменной, то функция вызывается так же, как и процедура – пишется имя этой функции и через пробел указывается список ее аргументов:

```
MyFunction 3, 5
```

Если необходимо записать значение функции в какую-либо переменную, то аргументы функции заключаются в круглые скобки:

```
Dim a
a = MyFunction(3, 5)
```

В заключение можно отметить, что приведенный объем справочного материала позволяет решить большинство из учебно-практических задач по программированию сценариев в среде VSH.

СПИСОК ЛИТЕРАТУРЫ

1. *Попов, А*. Windows Script Host для Windows 2000/XP / А. Попов. – БВХ-Петербург, 2005. – 320 с.

2. *Борн, Г.* Руководство разработчика на Microsoft Windows Script Host 2.0. / Г. Борн. – СПб. : Питер, Русская Редакция, 2001. – 479 с.

3. *Торрес, Д*. Скрипты для администратора Windows. Специальный справочник / Д. Торрес. – СПб. : Питер, 2002. – 351 с.

Интернет-ресурсы

4. http://support.microsoft.com/kb/188135

5. http://www.alexfl.ru/wsh/wsh.html

6. http://delphiregedit.narod.ru/regeditxp/

Губин Александр Николаевич
Филиппов Феликс Васильевич

ОСНОВЫ ПРОГРАММИРОВАНИЯ СЦЕНАРИЕВ В СРЕДЕ WSH

Учебное пособие

Редактор *Л. А. Медведева*

План 2013 г., п. 110

Подписано к печати 16.04.2013
Объем 6,0 усл.-печ. л. Тираж 40 экз. Заказ 313
Редакционно-издательский центр СПбГУТ.
191186 СПб., наб. р. Мойки, 61
Отпечатано в СПбГУТ