Андрей Попов

bhv®
www.bhv.ru
www.bhv.kiev.ua

Командные файлы и сценарии Windows Script Host

- Язык командных файлов Windows 9x/NT/2000
- Утилиты пакета Windows NT Resource Kit
- Стандартные объекты Windows Script Host

Мастер

УДК 681.3.06

Приводятся сведения об основных командах Windows 9*x*/NT/2000, а также об утилитах пакета Windows NT Resource Kit, необходимых при написании командных файлов. Описывается сервер сценариев Windows Script Host (WSH), в котором для автоматизации задач администрирования операционной системы и управления работой приложений могут применяться объекты ActiveX. Даны примеры командных файлов и сценариев WSH, написанных на языке JScript и осуществляющих доступ к файловой системе компьютера, базам данных и службам каталогов.

Для широкого круга пользователей

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Анатолий Адаменко*
Зав. редакцией	*Наталья Таркова*
Редактор	*Михаил Кокорев*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Попов А. В.

Командные файлы и сценарии Windows Script Host. — СПб.: БХВ-Петербург, 2002. — 320 с.

ISBN 5-94157-092-9

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 10 10.01.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л 25,8
Тираж 3000 экз. Заказ 1235
"БХВ-Петербург", 198005, Санкт-Петербург, Измаиловский пр., 29.

Гигиеническое заключение на продукцию, товар, № 77.99.1.953.П.950.3.99
от 01 03 1999 г выдано Департаментом ГСЭН Минздрава России

Отпечатано с готовых диапозитивов
в Академической типографии «Наука» РАН.
Качество соответствует предоставленным оригиналам.
199034,Санкт-Петербург,9 линия,12

Содержание

УДК 681.3.06

Приводятся сведения об основных командах Windows 9*x*/NT/2000, а также об утилитах пакета Windows NT Resource Kit, необходимых при написании командных файлов. Описывается сервер сценариев Windows Script Host (WSH), в котором для автоматизации задач администрирования операционной системы и управления работой приложений могут применяться объекты ActiveX. Даны примеры командных файлов и сценариев WSH, написанных на языке JScript и осуществляющих доступ к файловой системе компьютера, базам данных и службам каталогов.

Для широкого круга пользователей

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Анатолий Адаменко*
Зав. редакцией	*Наталья Таркова*
Редактор	*Михаил Кокорев*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Попов А. В.
Командные файлы и сценарии Windows Script Host. — СПб.: БХВ-Петербург, 2002. — 320 с.

ISBN 5-94157-092-9

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 10.10 01.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 25,8
Тираж 3000 экз. Заказ 1235
"БХВ-Петербург", 198005, Санкт-Петербург, Измаиловский пр., 29.

Гигиеническое заключение на продукцию, товар, № 77 99 1 953.П 950.3.99
от 01 03 1999 г. выдано Департаментом ГСЭН Минздрава России

Отпечатано с готовых диапозитивов
в Академической типографии «Наука» РАН.
Качество соответствует предоставленным оригиналам.
199034, Санкт-Петербург, 9 линия, 12

Содержание

Благодарности

Автор благодарит всех, без чьего содействия эта книга никогда не была бы написана. Спасибо сотрудникам Центра Интернет Мордовского госуниверситета и лично Эдуарду Батаршину и Михаилу Борисову за ценные консультации.

Выражаю признательность всем сотрудникам издательства "БХВ-Петербург", принимавшим участие в процессе создания книги, в особенности Анатолию Николаевичу Адаменко и Михаилу Сергеевичу Кокореву за предложения и замечания, значительно улучшившие книгу.

И наконец, отдельная благодарность моей жене Татьяне и другим родственникам за их терпение и поддержку во время работы над книгой.

Введение

Многие начинающие пользователи и администраторы операционной системы Windows, которым не приходилось ранее работать с MS-DOS, часто просто не подозревают о тех удобствах в повседневной работе, которые при грамотном применении могут им предоставить стандартные команды Windows и сценарии на базе командной строки (пакетные или командные файлы). Ведь ежедневные рутинные задачи (связанные, например, с копированием или архивированием файлов, подключением или отключением сетевых ресурсов и т. п.), которые обычно выполняются с помощью графического интерфейса проводника Windows, можно полностью автоматизировать, написав командный файл, состоящий всего из нескольких строчек. Более того, для человека, не знающего основные команды Windows и такие базовые возможности операционной системы, как перенаправление ввода/вывода и конвейеризация команд, некоторые простейшие задачи могут показаться очень трудными. Попробуйте, например, пользуясь только графическими средствами, сформировать файл, содержащий имена файлов из всех подкаталогов какого-либо каталога. А ведь для этого достаточно выполнить единственную команду `DIR` (с определенным ключом) и перенаправить вывод этой команды в нужный текстовый файл.

Очень ярко пренебрежение командной строкой проявляется при работе с Windows NT. Не все, к сожалению, знают, что интерпретатор команд в этой операционной системе имеет расширенный режим, который обеспечивает намного более богатые возможности при использовании командной строки и пакетных файлов. Автору практически не встречались книги на русском языке, где подробно описывались бы команды Windows 95/98 (далее Windows 9*x*) и Windows NT, пояснялись их отличия от команд MS-DOS, рассматривались особенности (весьма существенные) командных файлов в Windows NT вообще и сценариев регистрации пользователей в частности. Обычно в литературе описываются лишь несколько основных команд Windows ([1], [2], [7]) или приводится краткий список этих команд в приложениях ([4], [8]); в [3] и [4], правда, весьма подробно описаны некоторые

полезные утилиты командной строки из пакета Windows NT Resource Kit. Отметим, что за рубежом имеются книги, полностью посвященные рассмотрению команд и пакетных файлов, применяемых в Windows NT/2000 ([19], [20]), так что эта тема не может считаться "само собой разумеющейся" и слишком простой для раскрытия ее в отдельной книге.

Несколько лет назад компания Microsoft предложила в качестве инструмента разработки и выполнения специальных сценариев для операционной системы Windows сервер сценариев Windows Scripting Host (WSH), различные версии которого входят в стандартную поставку Windows 98 и Windows 2000. Сценарии WSH могут писаться на современных языках (например, Microsoft Visual Basic Script Edition (VBScript) или Microsoft JScript) и использовать любые объекты ActiveX, зарегистрированные в системе, что и определяет чрезвычайно мощные возможности таких сценариев. Надо сказать, что в то время как за рубежом сценариям WSH и используемым в них ActiveX-технологиям Microsoft посвящено довольно много серьезных и объемных книг ([16]—[18], [21]—[24]), на русском языке пока имеется только недавно вышедший перевод книги Т. Экка [9], посвященный сценариям Active Directory Service Interface (ADSI), и несколько публикаций в журналах ([10]—[15]). Примерно в такой же пропорции находится число зарубежных и российских сайтов, посвященных современным сценариям Windows (ссылки на соответствующие ресурсы сети Internet приведены в *приложении 5*).

Итак, целью настоящей книги является решение двух задач.

- Дать систематизированное описание работы с командной строкой в Windows, подробно разъясняя при этом различия между командами и пакетными файлами, использующимися в Windows 9*x* и Windows NT соответственно.
- Детально описать объектную модель WSH и привести практические примеры использования в сценариях WSH различных современных ActiveX-технологий компании Microsoft.

Отметим, что в Windows 2000 возможности командного интерпретатора остались практически теми же, что в Windows NT (незначительно изменены ключи у некоторых команд, добавлено несколько новых утилит командной строки), поэтому приведенное нами описание команд и пакетных файлов в целом годится и для Windows 2000.

Для кого предназначена эта книга

Сведения, изложенные в книге, могут быть полезны как начинающим пользователям и администраторам Windows 9*x*/NT/2000, которые хотят научиться грамотно работать с командной строкой и самостоятельно автоматизировать выполнение своих повседневных задач, так и профессионалам, желающим получить систематизированную информацию о мощных возможностях ActiveX-сценариев WSH, в том числе о сценариях с разметкой XML.

Кроме этого, книга может служить кратким справочникам по объектам, свойствам и методам WSH 2.0.

Первая часть книги, посвященная стандартным командам Windows, утилитам Windows NT Resource Kit и пакетным файлам, вполне доступна любому пользователю персонального компьютера, здесь не требуется никаких дополнительных знаний. При изучении же материала второй части, связанного со сценариями WSH, от читателя может потребоваться некоторое знакомство с алгоритмическими языками и общими понятиями объектно-ориентированного программирования. В книге будет приведен необходимый минимум сведений о языке Microsoft JScript, который использовался при написании всех примеров сценариев, вошедших в книгу (более подробная информация о JScript может быть найдена, например, в [5], [6]).

Структура книги

В *первой главе* рассмотрены основные принципы работы с командной строкой (командным интерпретатором `command.com`) в Windows 9*x*. Описанные здесь команды используются для решения следующих задач.

- Работа с файловой системой компьютера:
  - получение информации о структуре файловой системы;
  - просмотр или изменение атрибутов каталогов и файлов;
  - создание, копирование или удаление каталогов и файлов;
  - создание, изменение или удаление метки диска.
- Использование ресурсов локальной сети:
  - получение информации о доступных сетевых ресурсах;
  - подключение или отключение сетевых дисков и принтеров;
  - получение информации о доступных очередях печати;
  - синхронизация системного времени с другим компьютером;
- Просмотр и изменение переменных среды.
- Запуск в различных режимах внешних программ.
- Открытие документов зарегистрированных типов.

Также подробно рассмотрены утилиты командной строки, позволяющие искать информацию в текстовых или двоичных файлах, сортировать текстовые файлы и сравнивать содержимое двух файлов. Описание практически всех команд и утилит снабжено примерами их использования.

Кроме этого, в первой главе рассмотрены используемые в операционной системе механизмы переназначения устройств стандартного ввода/вывода и конвейеризации команд. С помощью этих механизмов можно, скажем, на-

править выводимую командами информацию не на экран, а на другое устройство (во внешний файл, на принтер и т. п.), или передать сообщения, выводимые одной программой, в качестве входных данных для другой программы (например, для сортировки "на лету" строк этих сообщений).

В отдельном разделе первой главы описаны процесс начальной загрузки Windows 9*x* и структура файлов `msdos.sys`, `config.sys` и `autoexec.bat`, использующихся для конфигурации операционной системы при загрузке.

Во *второй главе*, посвященной рассмотрению командных файлов, использующихся в Windows 9*x*, показано, как можно менять параметры выполнения пакетного файла. Также подробно описаны команды, позволяющие:

- выводить текстовую информацию на экран (или с помощью механизма перенаправления вывода на другое устройство компьютера);
- использовать внутри пакетного файла переменные среды и параметры командной строки, с которыми был запущен этот файл;
- запускать внешние командные файлы.

Кроме этого рассмотрено, каким образом можно управлять ходом выполнения командного файла с помощью циклов, безусловных переходов и операторов условий.

В одном из разделов второй главы описаны два вида меню, с помощью которых можно управлять процессом загрузки Windows. Подробно рассмотрены используемые в файлах `config.sys` и `autoexec.bat` команды, с помощью которых можно задать произвольное число различных конфигураций системы.

В конце второй главы приведено несколько упражнений на составление пакетных файлов для Windows 9*x*.

В *третьей главе* речь идет о внутренних командах Windows NT (практически все эти команды используются и в Windows 2000) и о командах пакета Windows NT Resource Kit. Основное внимание здесь уделено новым возможностям, которые предоставляет используемый в Windows NT командный интерпретатор `cmd.exe`:

- запуск программ и команд операционной системы в заданное время;
- ввод в командной строке сразу нескольких команд с возможностью управления запуском определенной команды в зависимости от результатов выполнения предыдущих (условное выполнение команд);
- поиск строк в текстовых файлах с использованием регулярных выражений и др.

Также в третьей главе подробно описаны нововведения в командах, предназначенных для работы с двумя файловыми системами, используемыми в Windows NT: FAT (File Allocation Table) и NTFS (New Technology File System).

В *четвертой главе* рассмотрены особенности и новые возможности командных файлов в Windows NT, описан процесс настройки параметров их выполнения. Приведены команды, позволяющие:

- производить арифметические действия над переменными среды и подставляемыми параметрами;
- выполнять циклы со счетчиком;
- считывать данные из текстового файла с разделителями.

Кроме того, в четвертой главе описаны некоторые утилиты Windows NT Resource Kit, которые могут оказаться весьма полезными при написании пакетных файлов, приведены примеры использования этих утилит. Отдельный раздел четвертой главы посвящен созданию и использованию сценариев входа, выполняющихся при регистрации пользователей в домене.

В конце четвертой главы приведено несколько упражнений на составление пакетных файлов с использованием как стандартных команд Windows NT, так и утилит Resource Kit.

В *пятой главе* мы переходим к изучению сервера ActiveX-сценариев WSH. Сначала здесь приведены общие сведения о технологии ActiveX фирмы Microsoft и кратко описан язык сценариев Microsoft JScript (все примеры сценариев в книге написаны на этом языке). Основная часть пятой главы посвящена рассмотрению собственных объектов WSH. Подробно описаны свойства и методы этих объектов, с помощью которых в сценариях можно:

- использовать внешние объекты ActiveX и ресурсы локальной сети;
- выводить информацию в стандартный выходной поток или в окно Windows;
- считывать данные из стандартного входного потока;
- получать доступ к специальным папкам Windows и системному реестру;
- создавать или изменять переменные среды и ярлыки Windows.

В *шестой главе* рассматриваются вопросы практического использования в сценариях WSH различных ActiveX-технологий, позволяющих получать полный доступ к файловой системе компьютера (объект `FileSystemObject`, FSO), работать с различными базами данных (технология ADO) и объектами локальных сетей разных производителей (технология ADSI). Приведено много примеров сценариев, с помощью которых можно решать различные практические задачи:

- манипулировать файлами с ведением протокола выполненных действий;
- выполнять выборку нужных данных из баз данных;
- создавать и удалять пользователей или группы пользователей в домене;
- просматривать и изменять информацию о пользователе;

- формировать списки всех групп в домене или всех пользователей в группе и т. д.

Также в шестой главе подробно рассмотрены сценарии WSH с разметкой XML, перечислены новые возможности сценариев такого формата, приведены примеры сценариев, реализующих эти возможности.

В конце шестой главы в качестве упражнений предложено написать сценарии на языке JScript, в которых нужно будет использовать как собственные объекты WSH, так и объекты FSO, ADO, ADSI и разметку XML.

В *приложениях* приведены список команд и утилит командной строки Windows (с указанием, в какой именно версии Windows используется та или иная команда), список ошибок, которые могут возникать при выполнении сценариев WSH, описаны различия в версиях WSH (от версии 1.0 до бета-версии 5.6). Кроме этого, в приложении приведены ответы к упражнениям (в виде полностью готовых к исполнению командных файлов и сценариев на языке JScript).

Принятые в книге соглашения

При описании команд и утилит командной строки Windows мы использовали стандартные соглашения. Названия параметров команд набраны курсивом, необязательные параметры или ключи заключены в квадратные скобки "[]", например:

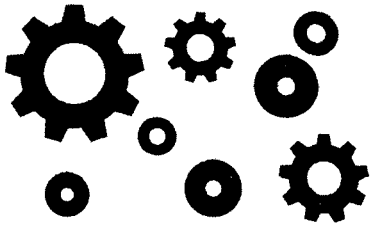
```
DEL [диск:][путь]имя_файла [/P]
```

Если при вызове команды может быть указан только один параметр/ключ из нескольких возможных, то такие параметры/ключи разделены знаком "|", например:

```
ATTRIB [+R|-R][+A|-A][+S|-S][+H|-H] [[диск:][путь]имя_файла] [/S]
```

Точно такие же обозначения для параметров (обязательных или необязательных) применены при описании функций языка JScript, методов объектов WSH или элементов XML.

ГЛАВА 1



Основные команды Windows 9*x*

В операционной системе Windows команды выполняются с помощью так называемого *командного интерпретатора* или *командного процессора*. Командный интерпретатор — это программа, которая, находясь в оперативной памяти, считывает набираемые вами команды и обрабатывает их. Некоторые команды распознаются и выполняются непосредственно самим командным интерпретатором — такие команды называются *внутренними*. Другие команды операционной системы представляют собой отдельные программы, расположенные на жестком диске, которые Windows загружает и выполняет аналогично другим программам. Такие команды называются *внешними*. Поэтому если у вас не запускается какая-нибудь внешняя команда, то это означает, что система не может найти соответствующий выполняемый файл.

Используемые в Windows 9*x* команды в основном являются доработанными командами MS-DOS 6.2. Необходимо отметить, что по разным причинам в Windows 9*x* были включены не все команды из MS-DOS, хотя некоторые из этих отсутствующих команд имеются в Windows NT. Сведения о наличии той или иной команды в Windows 9*x* и Windows NT приведены в *приложении 1*.

Из графической среды Windows доступ к командной строке, в которой вводятся команды для исполнения, можно получить несколькими способами.

Во-первых, можно выбрать пункт **Сеанс MS-DOS** (Command Prompt) в меню **Пуск/Программы** (Start/Programs). Фактически при этом в новом окне запускается командный процессор `command.com` (рис. 1.1).

Можно открыть несколько таких окон, которые мы будем называть *командными*. Если в вашем меню **Пуск/Программы** (Start/Programs) нет пункта **Сеанс MS-DOS** (Command Prompt), то можно просто выбрать пункт **Выполнить** (Run) и запустить программу `command.com`.

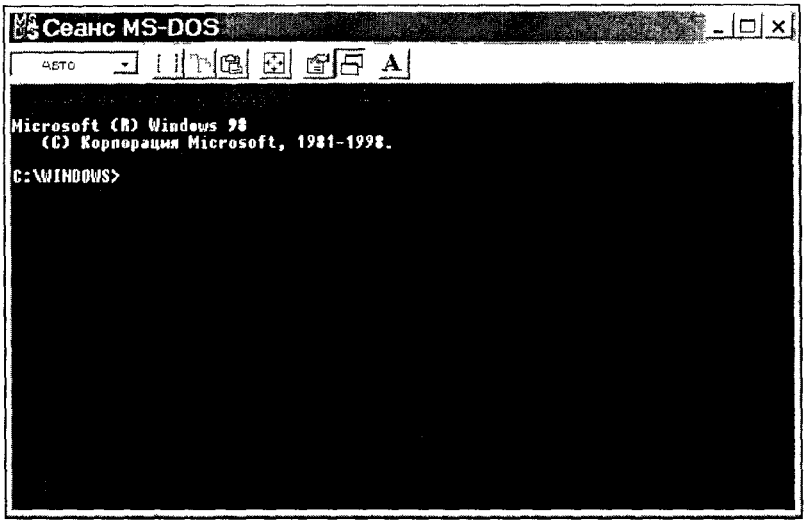


Рис. 1.1. Командное окно в Windows 98

Во-вторых, с помощью пункта меню **Пуск/Завершение работы** (Start/Shutdown) можно перезагрузить компьютер в режиме MS-DOS. Однако при этом вы уже не сможете пользоваться графической средой Windows. Этот режим работы в данной книге подробно рассматриваться не будет.

Наконец, *в-третьих*, можно запустить какую-нибудь программу-оболочку типа Norton Commander, Dos Navigator или Far (рис. 1.2).

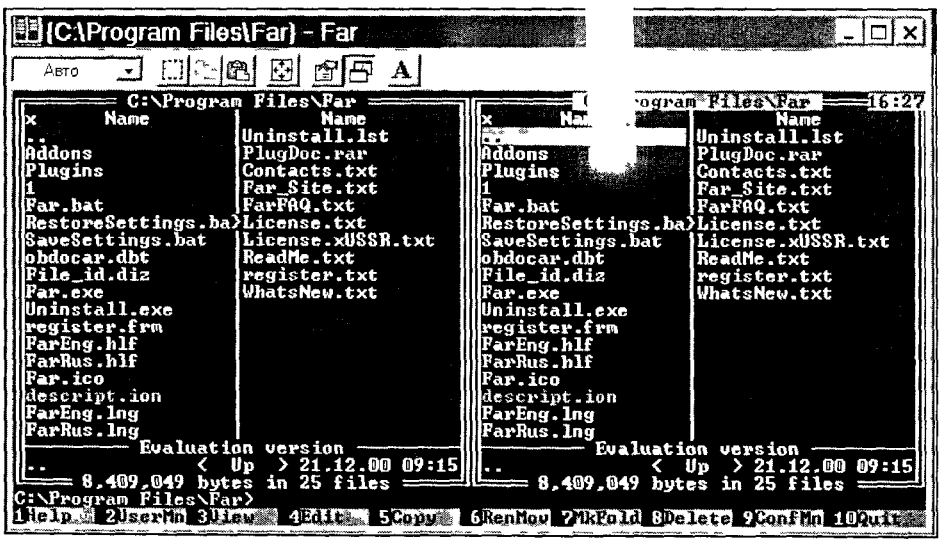


Рис. 1.2. Менеджер файлов и архивов Far

Очень удобной программой для работы с командной строкой в Windows является Far, т. к. она, обеспечивая привычный текстовый интерфейс, является полноценным 32-разрядным приложением Windows со множеством интересных возможностей.

Кроме этого, в Windows 98 можно в качестве командной строки пользоваться адресной строкой панели задач. Для этого приходится прибегать к помощи командных файлов, поэтому такая "адресно-командная" строка будет описана во второй главе.

Структура самой командной строки и принцип работы с ней в Windows остались теми же, что и в MS-DOS. Для того чтобы выполнить команду, вы после приглашения командной строки (например, `C:\>`) вводите имя этой команды (регистр не важен), ее параметры и ключи (если они необходимы) и нажимаете клавишу <Enter>. Например:

```
C:\>COPY C:\myfile.txt A:\ /V
```

Имя команды здесь — `COPY`, параметры — `C:\myfile.txt` и `A:\`, а ключом является `/V`.

Замечание

В некоторых командах ключи могут начинаться с символа - (минус), например, `-V`

Многие команды Windows имеют большое количество дополнительных параметров и ключей, запомнить которые зачастую бывает трудно. Большинство команд снабжено встроенной справкой, в которой кратко описываются назначение и синтаксис данной команды. Получить доступ к такой справке можно путем ввода команды с ключом `/?`. Например, если выполнить команду `ATTRIB /?`, то в окне MS-DOS мы увидим следующий текст:

```
Отображение и изменение атрибутов файлов.
ATTRIB [+R|-R] [+A|-A] [+S|-S] [+H|-H] [[диск:][путь]имя_файла] [/S]
  +   Установка атрибута.
  -   Снятие атрибута.
  R   Атрибут "Только чтение".
  A   Атрибут "Архивный".
  S   Атрибут "Системный".
  H   Атрибут "Скрытый".
  /S  Обработка файлов во всех вложенных папках указанного пути.
```

Для некоторых команд текст встроенной справки может быть довольно большим и не умещаться на одном экране. В этом случае помощь можно выводить последовательно по одному экрану с помощью команды `MORE` и символа конвейеризации `|`, например:

```
XCOPY /? | MORE
```

В этом случае после заполнения очередного экрана вывод помощи будет прерываться до нажатия любой клавиши. Кроме того, используя символы > и >> перенаправления вывода, можно текст, выводимый на экран, направить в текстовый файл для дальнейшего просмотра. Например, для вывода текста справки к команде XCOPY в текстовый файл xcopy.txt, используется следующая команда:

```
XCOPY /? > XCOPY.TXT
```

Подробнее команда MORE и перенаправление выводимого на экран текста в файл будут описаны далее в *разд. "Перенаправление ввода/вывода и конвейеризация команд"* этой главы.

Отметим, что для сетевых команд Windows (они начинаются со слова NET) доступно еще несколько вариантов встроенной помощи (табл. 1.1).

Таблица 1.1. *Вызов встроенной справки к сетевым командам*

Синтаксис	Описание
NET HELP	Список всех доступных сетевых команд
NET *команда* /?	Назначение, синтаксис, параметры и ключи команды. Например: NET USE /?
NET HELP *команда*	То же, что NET *команда* /? Например: NET HELP USE
NET *команда* HELP	То же, что NET команда /? Например: NET USE HELP

Перейдем теперь к подробному описанию основных команд, использующихся в Windows 9*x*.

Командный интерпретатор *command.com*

В Windows 9*x* командный интерпретатор по умолчанию представлен исполняемым файлом command.com, который находится в системном каталоге Windows, содержащем файлы для загрузки операционной системы. Этот системный каталог задается значением переменной WinBootDir в разделе [Paths] конфигурационного файла msdos.sys, находящегося в корневом каталоге загрузочного диска (подробнее структура этого файла описана ниже в *разд. "Команды конфигурации системы"* данной главы). Например,

```
[Paths]
WinBootDir=C:\Windows
```

Далее в примерах для определенности будем считать, что системным каталогом является C:\WINDOWS.

Как уже отмечалось выше, для запуска command.com (т. е. для выполнения команды COMMAND) можно выбрать команду **Сеанс MS-DOS** (Command Prompt) в меню **Пуск/Программы** (Start/Programs) или просто воспользоваться командой **Выполнить** (Run) меню **Пуск** (Start). Загрузив тем или иным способом command.com, мы получаем новое командное окно (окно MS-DOS), в котором можно пользоваться командной строкой, т. е. вызывать другие внутренние и внешние команды.

Внутренними командами для интерпретатора command.com являются следующие команды Windows: BREAK, CALL, CHCP, CHDIR, CLS, COPY, CTTY, DATE, DEL, DIR, ECHO, EXIT, FOR, GOTO, IF, LOADHIGH, MKDIR, PATH, PAUSE, PROMPT, REM, RENAME, RMDIR, SET, SHIFT, TIME, TYPE, VER, VERIFY, VOL. Все остальные команды являются внешними и представлены исполняемыми файлами; эти файлы после установки операционной системы находятся в папке C:\WINDOWS\COMMAND.

Отметим, что при запуске команды COMMAND можно при необходимости указать ряд дополнительных параметров и ключей. Синтаксис команды COMMAND имеет вид:

```
COMMAND [[диск:]путь] [устройство] [/E:nnnnn] [/L:nnnn] [/U:nnn] [/P]
[/MSG] [/LOW] [/Y [/[C|K] команда]]
```

Краткое описание используемых параметров и ключей для команды COMMAND приведено в табл. 1.2.

Таблица 1.2. *Параметры и ключи команды* COMMAND

Параметр	Описание
[*диск*:]*путь*	Каталог для поиска файла command.com. Этот параметр может понадобиться при перезагрузке нерезидентной части командного интерпретатора и должен указываться в том случае, когда command.com находится в каталоге, отличном от C:\WINDOWS
устройство	Устройство, использующееся для вывода и ввода команд. Для задания этого устройства также можно использовать команду CTTY
/E:nnnnn	Исходный размер области переменных среды в байтах. (Число nnnnn должно лежать в пределах от 256 до 32 768)
/L:nnnn	Длина внутренних буферов (требуется ключ /P). (Число nnnnn должно лежать в пределах от 128 до 1024)
/U:nnn	Длина буфера ввода (требуется ключ /P). (Число nnnnn должно лежать в пределах от 128 до 255)
/P	Загрузка интерпретатора команд без возможности выхода из него

***Таблица 1.2** (окончание)*

Параметр	Описание
/MSG	Хранение сообщений об ошибках в памяти (требуется ключ /P)
/LOW	Размещение резидентных данных COMMAND в нижней области памяти
/Y	Выполнение пакетного файла, определенного ключом /C или /K, по шагам
/C *команда*	Исполнение указанной команды и возврат. Этот параметр должен быть последним в командной строке
/K *команда*	Исполнение указанной команды и продолжение выполнения. Этот параметр должен быть последним в командной строке

Используя ключи /C и /K можно, скажем, выполнять внутренние команды Windows с помощью команды **Выполнить** (Run) из меню **Пуск** (Start), а также создавать ярлыки для выполнения внутренних команд, как показано на рис. 1.3.

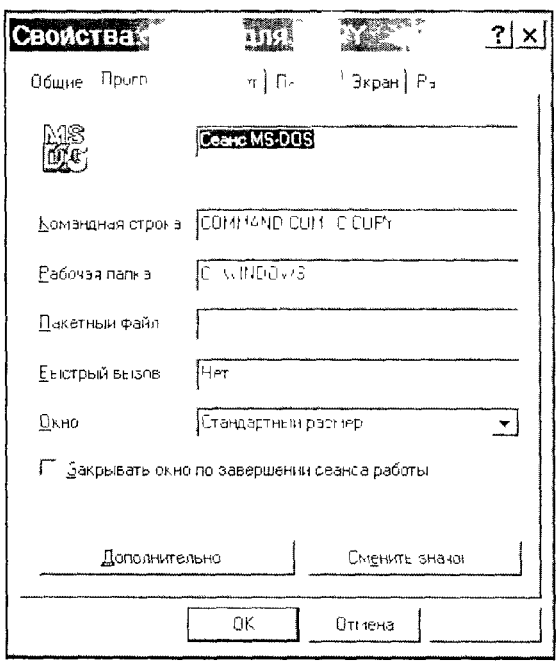


Рис. 1.3. Ярлык для внутренней команды COPY

Ключ /Y может быть полезен для отладки пакетных файлов (подробнее это будет обсуждаться во второй главе).

В уже открытом командном окне можно запустить новую копию интерпретатора команд Windows, вновь введя в командной строке команду COMMAND (новое окно при этом не открывается). Для остановки нового командного интерпретатора и возврата в старый используется команда EXIT. Если же в командном окне запущена только одна копия интерпретатора, то выполнение команды EXIT приведет к закрытию этого окна.

Команды для работы с файловой системой

В данном разделе будут подробно рассмотрены следующие команды: CD, ATTRIB, COPY, XCOPY, DIR, MKDIR, RMDIR, DEL, DELTREE, REN, MOVE, SUBST, VOL, LABEL.

Отметим сначала несколько особенностей определения путей к файлам в Windows. Напомним, что файловая система логически имеет древовидную структуру и имена файлов задаются в формате [*диск*:][*путь*\]*имя_файла*, т. е. обязательным параметром является только имя файла. При этом, если путь начинается с символа "\", то маршрут вычисляется от корневого каталога, иначе — от текущего каталога. Например, имя C:123.txt задает файл 123.txt в *текущем* каталоге на диске C:, имя C:\123.txt — файл 123.txt в *корневом* каталоге на диске C:, имя ABC\123.txt — файл 123.txt в подкаталоге ABC текущего каталога.

Существуют особые обозначения для текущего каталога и трех его верхних уровней. Текущий каталог обозначается символом . (точка), его родительский каталог — символами .. (две точки), каталог второго уровня — символами ... (три точки), и, наконец, каталог третьего уровня — символами .... (четыре точки). Например, если текущим каталогом является C:\WINDOWS\MEDIA\OFFICE97, то путь к файлу autoexec.bat в корневом каталоге диска C: может быть записан в виде ....\autoexec.bat.

В именах файлов (но не дисков или каталогов) можно применять так называемые *групповые символы* или *шаблоны*: ? (вопросительный знак) и * (звездочка). Символ * в имени файла означает произвольное количество любых допустимых символов, символ ? — один произвольный символ или его отсутствие. Скажем, под шаблон text??1.txt подходят, например, имена text121.txt и text11.txt, под шаблон text*.txt — имена text.txt, textab12.txt, а под шаблон text.* — все файлы с именем text и произвольным расширением.

Как известно, в Windows 9*x* можно использовать длинные имена файлов, что снимает имевшиеся в MS-DOS ограничения на длину имени файла и символы, которые можно использовать в этом имени. Для того чтобы ис-

пользовать длинные имена файлов при работе с командной строкой, их нужно заключать в двойные кавычки. Например, чтобы запустить файл с именем Мое приложение.exe из каталога Мои документы, нужно в командной строке набрать C:\Мои документы\Мое приложение.exe и нажать клавишу <Enter>.

Перейдем теперь непосредственно к командам для работы с файловой системой.

Команда *CD*

Текущий каталог можно изменить с помощью команды

```
CD [диск:][путь\]
```

Если запустить команду CD без параметров, то на экран будут выведены имена текущего диска и каталога. Приведем несколько примеров использования команды CD.

Пусть у нас имеется представленная на рис. 1.4 структура каталогов, причем текущим является каталог BOOKS.

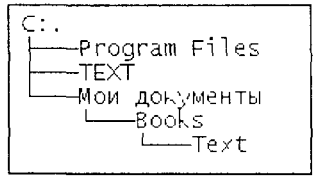


Рис. 1.4. Пример дерева каталогов

С помощью команды CD можно выполнить следующие действия.

1. Переход в корневой каталог текущего диска:

   ```
   CD \
   ```

2. Переход из каталога BOOKS в подкаталог TEXT текущего каталога:

   ```
   CD TEXT
   ```

3. Переход из каталога BOOKS в каталог C:\TEXT текущего диска:

   ```
   CD \TEXT
   ```

4. Переход из каталога BOOKS в каталог Мои документы:

   ```
   CD "\Мои документы"
   ```

   или

   ```
   CD.
   ```

Команда *ATTRIB*

В операционных системах Windows 9*x*, как и в MS-DOS, используется файловая система, основанная на FAT (File Allocation Table, таблица размещения файлов). В этой файловой системе каждый файл или каталог имеет набор некоторых свойств, называемых атрибутами. Файл может иметь следующие атрибуты: "Только для чтения" (Read-Only), "Системный" (System), "Архивный" (Archive) и "Скрытый" (Hidden). В Windows атрибуты файла или группы файлов можно изменять с помощью Проводника (Explorer). Для этого необходимо выделить нужные файлы, щелкнуть правой кнопкой мыши и выбрать пункт **Свойства** (Properties) в контекстном меню. В появившемся окне (рис. 1.5) можно производить необходимые изменения, при этом отметим, что таким образом у файла нельзя изменить атрибут "Системный".

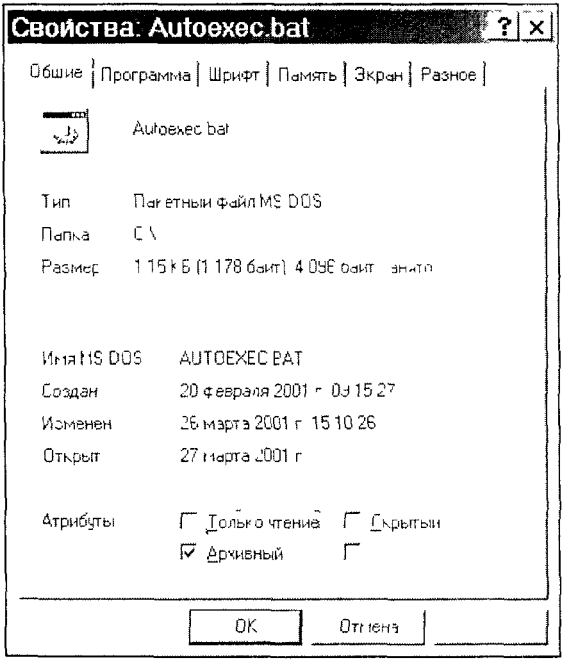


Рис. 1.5. Изменение атрибутов файла в Проводнике Windows

Для просмотра и изменения атрибутов файла или группы файлов с помощью командной строки используется команда ATTRIB.

Синтаксис этой команды:

```
ATTRIB [+R|-R] [+A|-A] [+S|-S] [+H|-H] [[диск:] [путь] имя_файла] [/S]
```

Здесь ключи + и - означают соответственно установку или снятие соответствующего атрибута: R — "Только чтение", A — "Архивный", S — "Системный", H — "Скрытый".

Замечание

Атрибуты файлов можно посмотреть и с помощью команды DIR с ключом /V *(см. ниже)*.

Ключ /S указывает на то, что команда ATTRIB должна обработать файлы во всех подкаталогах указанного пути.

Замечание

Вообще, ключ /S во всех командах, в которых он встречается, указывает на то, что необходимо произвести какие-либо действия с файлами не только в каталоге по указанному пути, но и во всех его подкаталогах. Этот ключ может значительно упростить групповую обработку файлов, т. к. пропадает необходимость выполнять требуемую команду для каждого отдельного подкаталога.

Рассмотрим примеры использования команды ATTRIB.

1. Сделать все файлы в каталоге C:\PROGRAMS доступными только для чтения, а сам этот каталог — скрытым:

```
ATTRIB +R C:\PROGRAMS\*.*
ATTRIB +H C:\PROGRAMS
```

2. Требуется скрыть на диске C: все файлы текущего каталога и его подкаталогов, кроме файлов с расширением bak. Для этого можно воспользоваться двумя командами:

```
ATTRIB +H C:*.* /S
ATTRIB -H C:*.bak /S
```

Команда *COPY*

Одной из наиболее часто повторяющихся задач при работе на компьютере является копирование и перемещение файлов из одного места в другое. Для копирования одного или нескольких файлов используется команда COPY.

Синтаксис этой команды:

```
COPY [/A|/B] источник [/A|/B] [+ источник [/A|/B] [+ ...]]
  [результат [/A|/B]] [/V][/Y|/-Y]
```

Краткое описание используемых параметров и ключей этой команды приведено в табл. 1.3.

Таблица 1.3. *Параметры и ключи команды* COPY

Параметр	Описание
источник	Имя копируемого файла или файлов
/A	Файл является текстовым файлом ASCII, т. е. конец файла обозначается символом с кодом ASCII 26 (<Ctrl>+<Z>)
/B	Файл является двоичным. Этот ключ указывает на то, что интерпретатор команд должен при копировании считывать из источника число байтов, заданное размером в каталоге копируемого файла
результат	Каталог для размещения результата копирования и/или имя создаваемого файла
/V	Проверка правильности копирования путем сравнения файлов после копирования
/Y	Отключение режима запроса подтверждения на замену файлов
/-Y	Включение режима запроса подтверждения на замену файлов

Приведем примеры использования команды COPY.

1. Копирование файла abc.txt из текущего каталога в каталог D:\PROGRAM под тем же именем:

```
COPY abc.txt D:\PROGRAM
```

2. Копирование файла abc.txt из текущего каталога в каталог D:\PROGRAM под новым именем def.txt:

```
COPY abc.txt D:\PROGRAM\def.txt
```

3. Копирование всех файлов с расширением txt с диска A: в каталог Мои документы на диске C:

```
COPY A:\*.txt "C:\Мои документы"
```

4. Если не задать в команде целевой файл, то команда COPY создаст копию файла-источника с тем же именем, датой и временем создания, что и исходный файл, и поместит новую копию в текущий каталог на текущем диске. Например, для того чтобы скопировать все файлы из корневого каталога диска A: в текущий каталог, достаточно выполнить такую краткую команду:

```
COPY A:\*.*
```

Действие ключей /A и /B зависит от их позиции в командной строке. Если эти параметры следуют за именем исходного файла, то COPY работает следующим образом. Задание ключа /A приводит к интерпретации файла как файла ASCII (текстовый файл) и копированию только данных, предшест-

вующих первому символу конца файла. При этом команда COPY не копирует сам символ конца файла и остаток файла. Задание же ключа /B приводит к копированию всего файла, включая символ конца файла.

Когда ключи /A или /B следуют за именем целевого файла, COPY выполняет следующие действия. При задании /A в скопированный файл в качестве последнего символа добавляется символ конца файла. Например, команда

```
COPY abc.txt def.txt /A
```

выполнит копирование файла abc.txt в файл def.txt и добавит в def.txt символ конца файла. Если же в команде COPY указан ключ /B, то символ конца файла в скопированный файл не добавляется.

В качестве источника или результата при копировании можно указывать имена не только файлов, но и устройств компьютера. В Windows поддерживаются следующие имена устройств: PRN (принтер), LPT1 – LPT3 (соответствующие параллельные порты), AUX (устройство, присоединяемое к последовательному порту 1), COM1 – COM3 (соответствующие последовательные порты), CON (терминал: при вводе это клавиатура, при выводе — монитор), NUL (пустое устройство, все операции ввода/вывода для него игнорируются). Например, для того чтобы распечатать файл abc.txt на принтере, можно воспользоваться командой копирования этого файла на устройство PRN:

```
COPY abc.txt PRN
```

Другой интересный пример: создадим новый текстовый файл и запишем в него информацию, без использования текстового редактора. Для этого достаточно ввести команду

```
COPY CON my.txt
```

которая будет копировать то, что вы набираете на клавиатуре, в файл my.txt (если этот файл существовал, то он перезапишется, иначе — создастся). Для завершения ввода необходимо ввести символ конца файла, т. е. нажать клавиши <Ctrl>+<Z>.

Когда в качестве приемника выступает устройство компьютера, задание ключа /B приводит к тому, что Windows копирует данные на устройство в двоичном режиме, в котором все символы, включая специальные (такие как <Ctrl>+<C>, <Ctrl>+<S>, <Ctrl>+<Z> и возврат каретки), копируются на устройство как данные. Отсутствие ключа /B вызывает копирование данных на устройство в режиме ASCII. При этом перечисленные выше специальные символы вызывают при копировании особые действия (например, перевод каретки на принтере).

Команда COPY может также объединять (склеивать) несколько файлов в один. Для этого необходимо указать единственный результирующий файл и несколько исходных. Это достигается путем использования групповых зна-

ков (? и *) или формата файл1 + файл2 + файл3. Например, для объединения файлов 1.txt и 2.txt в файл 3.txt можно задать следующую команду:

```
COPY 1.txt+2.txt 3.txt
```

Объединение всех файлов с расширением dat из текущего каталога в один файл all.dt может быть произведено так:

```
COPY /B *.dat all.dt
```

Ключ /B здесь используется для предотвращения усечения соединяемых файлов, т. к. при комбинировании файлов команда COPY по умолчанию считает файлы текстовыми.

Если имя целевого файла совпадает с именем одного из копируемых файлов (кроме первого), то исходное содержимое целевого файла теряется. Если имя целевого файла опущено, то в его качестве используется первый файл из списка. Например, команда

```
COPY 1.txt+2.txt
```

добавит к содержимому файла 1.txt содержимое файла 2.txt. Командой COPY можно воспользоваться и для присвоения какому-либо файлу текущей даты и времени без модификации его содержимого. Для этого нужно ввести команду типа

```
COPY /B 1.txt +,,
```

Здесь запятые указывают на пропуск параметра приемника, что и приводит к требуемому результату.

Отметим теперь ряд недостатков команды COPY. Во-первых, с ее помощью нельзя копировать скрытые и системные файлы, файлы нулевой длины, файлы из подкаталогов. Во-вторых, используя COPY, нельзя при копировании заменять файлы с атрибутом "Только для чтения", что часто бывает необходимо. Наконец, если при копировании группы файлов команда COPY встретит файл, который в данный момент нельзя скопировать (например, он занят другим приложением), то процесс копирования полностью прервется, и остальные файлы не будут скопированы.

Команда *XCOPY*

Указанные проблемы можно решить с помощью команды XCOPY, которая предоставляет намного больше возможностей при копировании. Необходимо отметить, правда, что XCOPY может работать только с файлами и каталогами, но не с устройствами.

Синтаксис этой команды:

```
XCOPY источник [результат] [/A|/M][/D[:дата]][/P][/S[/E]][/W][/C][/I]
[/Q][/F][/L][/H][/R][/T][/U][/K][/Y|/-Y][/N]
```

Ключи /A и /M здесь определяют режим копирования файлов с установленным архивным атрибутом. Если указан ключ /A, то у таких файлов после копирования атрибут "Архивный" остается установленным, если ключ /M — атрибут "Архивный" после копирования снимается.

Ключ /D позволяет копировать только файлы, измененные не ранее указанной даты. Если параметр *дата* не указан, то копирование будет производиться, только если источник новее результата. Например, команда

```
XCOPY "C:\Мои документы\*.*" "D:\BACKUP\Мои документы" /D
```

скопирует в каталог D:\BACKUP\Мои документы только те файлы из каталога C:\Мои документы\, которые были изменены со времени последнего подобного копирования или которых вообще не было в D:\BACKUP\Мои документы.

Если указан ключ /P, то перед созданием файлов при копировании будет запрашиваться подтверждение.

Ключ /S позволяет копировать все непустые подкаталоги в каталоге-источнике. С помощью же ключа /E можно копировать вообще все подкаталоги, включая и пустые.

При помощи ключа /W устанавливается режим запроса на нажатие любой клавиши перед началом копирования.

Если указан ключ /C, то копирование будет продолжаться даже в случае возникновения ошибок. Это бывает очень полезным при операциях копирования, производимых над группами файлов, например, при резервном копировании данных.

Ключ /I важен для случая, когда копируются несколько файлов, а файл назначения отсутствует. При задании этого ключа команда XCOPY считает, что файл назначения должен быть каталогом. Например, если задать ключ /I в команде копирования всех файлов с расширением txt из текущего каталога в несуществующий еще подкаталог TEXT:

```
XCOPY  *.txt TEXT /I
```

то подкаталог TEXT будет создан без дополнительных запросов.

Ключи /Q, /F и /L отвечают за режим отображения при копировании. При задании ключа /Q имена файлов при копировании не отображаются, ключа /F — отображаются полные пути источника и результата. Ключ /L обозначает, что отображаются только файлы, которые должны быть скопированы (при этом само копирование не производится).

С помощью ключа /H можно копировать скрытые и системные файлы, а с помощью ключа /R — заменять файлы с атрибутом "Только для чтения". Например, для копирования всех файлов из корневого каталога диска C: (включая системные и скрытые) в каталог SYS на диске D: нужно ввести следующую команду:

```
XCOPY C:\*.* D:\SYS /H
```

Ключ /T позволяет применять команду XCOPY для копирования только структуры каталогов источника, без дублирования находящихся в этих каталогах файлов, причем пустые каталоги и подкаталоги не включаются. Для того чтобы все же включить пустые каталоги и подкаталоги, нужно использовать комбинацию ключей /T /E.

Используя XCOPY, можно при копировании обновлять только уже существующие файлы (новые файлы при этом не записываются). Для этого применяется ключ /U. Например, если в каталоге C:\2 находились файлы a.txt и b.txt, а в каталоге C:\1 — файлы a.txt, b.txt, c.txt и d.txt, то после выполнения команды

```
XCOPY C:\1 C:\2 /U
```

в каталоге C:\2 по-прежнему останутся лишь два файла: a.txt и b.txt, содержимое которых будет заменено содержимым соответствующих файлов из каталога C:\1. Если с помощью XCOPY копировался файл с атрибутом "Только для чтения", то по умолчанию у файла-копии этот атрибут снимется. Для того чтобы копировать не только данные, но и полностью атрибуты файла, необходимо использовать ключ /K.

Ключи /Y и /-Y определяют, нужно ли запрашивать подтверждение перед заменой файлов при копировании. /Y означает, что такой запрос нужен, /-Y — не нужен.

Наконец, ключ /N указывает, что при копировании файлов необходимо использовать короткие имена.

Все указанные ключи можно комбинировать друг с другом в произвольной последовательности. Например, для копирования всех файлов и подкаталогов (включая пустые) с диска A: на диск D:, с предварительным запросом перед началом копирования, можно задать следующую команду:

```
XCOPY A: D: /S /E /W
```

При использовании команды XCOPY внутри пакетных файлов можно анализировать ее *код выхода (код завершения)*, который определяет причину завершения выполнения команды, и в зависимости от этого кода предпринимать те или иные действия *(см. разд. "Переходы и операторы условия в командных файлах" гл. 2)*. Коды выхода команды XCOPY перечислены в табл. 1.4.

Таблица 1.4. *Коды завершения команды XCOPY*

Код	Описание
0	Файлы скопированы без ошибки
1	Файлы для копирования не найдены

Таблица 1.4 *(окончание)*

Код	Описание
2	Для прерывания XCOPY пользователь нажал <Ctrl>+<C>
4	Ошибка инициализации (на хватает памяти или места на диске, либо введено недопустимое имя диска или неверен синтаксис команды)
5	Ошибка записи на диск

Команда *DIR*

Еще одной очень полезной командой является DIR, которая используется для вывода информации о содержимом дисков и каталогов. Если задать эту команду без параметров, то выводится метка диска и его серийный номер, имена (в коротком и длинном вариантах) файлов и подкаталогов, находящихся в текущем каталоге, а также дата и время их последней модификации. После этого выводится число файлов в каталоге, общий объем (в байтах), занимаемый файлами, и объем свободного пространства на диске. Например:

```
Том в устройстве C имеет метку PHYS1_PART2
 Серийный номер тома: 366D-6107
 Содержимое папки C:\aditor
.              <ПАПКА>      25.01.00  17:15 .
..             <ПАПКА>      25.01.00  17:15 ..
TEMPLT02 DAT           227  07.08.98   1:00 templt02.dat
UNINST1  000         1 093  02.03.99   8:36 UNINST1.000
HILITE   DAT         1 082  18.09.98  18:55 hilite.dat
TEMPLT01 DAT            48  07.08.98   1:00 templt01.dat
UNINST0  000        40 960  15.04.98   2:08 UNINST0.000
TTABLE   DAT           357  07.08.98   1:00 ttable.dat
ADITOR   EXE       461 312  01.12.99  23:13 aditor.exe
README   TXT         3 974  25.01.00  17:26 readme.txt
ADITOR   HLP        24 594  08.10.98  23:12 aditor.hlp
TEKCTO~1 TXT             0  11.03.01   9:02 Текстовый файл.txt
        11 файлов        533 647 байт
         2 папок     143 261 696 байт свободно
```

С помощью ключей команды DIR можно задать различные режимы расположения, фильтрации и сортировки выводимой информации.

Синтаксис команды:

```
DIR [диск:][путь][имя_файла] [/P][/W][/A[[:]атрибуты]][/O[[:]сортировка]]
[/S][/B][/L][/V][/4]
```

Параметр [*диск*:][*путь*] задает диск и каталог, содержимое которого надо вывести на экран. Параметр [*имя_файла*] задает файл или группу файлов, которые нужно включить в список. Например, команда

```
DIR C:\*.bat
```

выведет на экран все файлы с расширением bat в корневом каталоге диска C:.

При указании ключа /P список файлов и подкаталогов выводится по одному экрану. Чтобы увидеть следующий экран, нужно нажать любую клавишу.

При использовании ключа /W перечень файлов выводится в широком формате с максимально возможным числом имен файлов или каталогов на каждой строке. Например:

```
Том в устройстве C имеет метку PHYS1_PART2
 Серийный номер тома: 366D-6107
 Содержимое папки C:\aditor
[.]             [..]            TEMPLT02.DAT   UNINST1.000    HILITE.DAT
TEMPLT01.DAT    UNINST0.000     TTABLE.DAT     ADITOR.EXE     README.TXT
ADITOR.HLP      TEKCTO~1.TXT
        11 файлов          533 647 байт
         2 папок       143 257 600 байт свободно
```

С помощью ключа /A[[:]*атрибуты*] можно вывести имена только тех каталогов и файлов, которые имеют заданные атрибуты. Атрибуты обозначаются теми же символами, что и в команде ATTRIB, префикс "–" имеет значение НЕ. Если ключ /A используется более чем с одним значением атрибута, будут выведены имена только тех файлов, у которых все атрибуты совпадают с заданными (значения атрибутов не разделяются пробелами). Например, для вывода имен всех файлов в корневом каталоге диска C:, которые одновременно являются скрытыми и системными, можно задать команду

```
DIR C:\ /A:HS
```

а для вывода всех файлов, кроме скрытых — команду

```
DIR C:\ /A:-H
```

Отметим здесь, что атрибуту каталога соответствует буква D, т. е. для того, чтобы, например, вывести список всех каталогов диска C:, нужно задать команду

```
DIR C: /A:D
```

Если ключ /A опущен, то команда DIR выводит все файлы, кроме скрытых и системных. Если же /A задан без указания атрибутов, то DIR выводит имена всех файлов, включая скрытые и системные.

Ключ /O[[:]*сортировка*] задает порядок сортировки содержимого каталога при выводе его командой DIR. Если этот ключ опущен, DIR печатает имена

файлов и каталогов в том порядке, в котором они содержатся в каталоге. Если ключ /O задан, а параметр *сортировка* не указан, то DIR выводит имена в алфавитном порядке. В параметре *сортировка* можно использовать следующие значения: N — по имени (алфавитная), S — по размеру (начиная с меньших), E — по расширению (алфавитная), D — по дате (начиная с более старых), A — по дате загрузки (начиная с более старых), G — начать список с каталогов. Префикс "-" означает обратный порядок. Если задается более одного значения порядка сортировки, файлы сортируются по первому критерию, затем по второму и т. д. При комбинировании параметров их не нужно разделять пробелами.

Ключ /S означает, как обычно, вывод списка файлов из заданного каталога и его подкаталогов.

Ключ /B перечисляет только названия каталогов и имена файлов (в длинном формате) по одному на строку, включая расширение. При этом выводится только основная информация, без итоговой. Например:

```
templt02.dat
UNINST1.000
hilite.dat
templt01.dat
UNINST0.000
ttable.dat
aditor.exe
readme.txt
aditor.hlp
Текстовый файл.txt
```

Отметим, что /B переопределяет ключ /W.

Ключ /L выводит неотсортированные имена файлов и каталогов в нижнем регистре.

С помощью ключа /V можно отобразить расширенные сведения о файлах и каталогах. При использовании этого ключа на экран выводятся: метка и серийный номер диска, короткое и длинное имя файла или каталога (в двух строчках), размер файла в байтах, реальный размер занятого файлом пространства на диске, дата последнего изменения файла или каталога, дата загрузки файла или каталога, атрибуты файла или каталога, общие количество и размер файлов и каталогов, количество занятого и свободного места на диске, заполнение диска в процентах.

Наконец, ключ /4 задает вывод четырех цифр года (если не указан ключ /V).

Часто бывает необходимо составлять отчеты, в которых требуется указывать информацию о файлах в заданных каталогах. Для этого данные, выводимые командой DIR, можно распечатать на принтере или перенаправить в текстовый файл, используя символы перенаправления > и >>.

Например:

```
DIR>PRN
```

или

```
DIR>list.txt
```

Подробнее перенаправление ввода/вывода будет описано далее в разд. *"Перенаправление ввода/вывода и конвейеризация команд"* этой главы.

Замечание

К сожалению, в Windows 9x не включена весьма полезная команда TREE из MS-DOS, с помощью которой можно было графически представлять древовидную структуру каталогов. Эта команда присутствует в Windows NT.

Приведем примеры использования команды DIR с различными комбинациями ключей.

1. Вывод содержимого всех подкаталогов каталога C:\Мои документы, причем названия подкаталогов и файлов упорядочиваются по алфавиту, выводятся в широком формате и после вывода каждого экрана делается пауза:

   ```
   DIR "C:\Мои документы" /S/W/O/P
   ```

2. Распечатка информации, аналогичной первому примеру, но без содержимого подкаталогов, на принтере:

   ```
   DIR "C:\Мои документы" /W/O/P > PRN
   ```

Команды *MKDIR, RMDIR*

Для создания нового каталога и удаления уже существующего пустого каталога используются команды MKDIR [*диск:*]*путь* и RMDIR [*диск:*]*путь* соответственно (или их короткие аналоги MD и RD). Например:

```
MKDIR "C:\Примеры"
RMDIR "C:\Примеры"
```

Команда MKDIR не может быть выполнена, если каталог или файл с заданным именем уже существует. Команда RMDIR не будет выполнена, если удаляемый каталог не пустой.

Команды *DEL, ERASE, DELTREE*

Удалить один или несколько файлов можно с помощью команд

```
DEL [диск:][путь]имя_файла [/P]
```

или

```
ERASE [диск:][путь]имя_файла [/P]
```

Для удаления сразу нескольких файлов используются групповые знаки ? и *. Ключ /P означает выдачу запросов при удалении каждого файла. Задание команды удаления всех файлов, DEL *.*, приводит к выводу на экран предварительного запроса перед выполнением команды.

Для удаления каталога вместе со всеми подкаталогами и файлами, которые в нем содержатся, используется команда

```
DELTREE [/Y] [диск:]путь [[диск:]путь[...]]
```

По умолчанию перед выполнением команды DELTREE на экран выводится запрос на подтверждение удаления. Отключить этот запрос можно с помощью ключа /Y.

Команды *RENAME, MOVE*

Переименовать файлы и каталоги можно с помощью команды RENAME (REN). Синтаксис этой команды имеет следующий вид:

```
REN [диск:][путь]каталог1|файл1 [каталог2|файл2]
```

Здесь параметр *каталог1|файл1* определяет название каталога/файла, которое нужно изменить, а *каталог2|файл2* задает новое название каталога/файла. В любом параметре команды REN можно использовать групповые символы ? и *. При этом представленные шаблонами символы в параметре *файл2* будут идентичны соответствующим символам в параметре *файл1*. Например, чтобы изменить у всех файлов с расширением txt в текущем каталоге расширение на doc, нужно ввести такую команду:

```
REN *.txt *.doc
```

Если файл с именем *файл2* уже существует, то команда REN прекратит выполнение, и произойдет вывод сообщения, что файл уже существует или занят. Кроме того, в команде REN нельзя указать другой диск или каталог для создания результирующих каталога и файла. Для этой цели нужно использовать команду MOVE, предназначенную для переименования и перемещения файлов и каталогов.

Синтаксис команды для перемещения одного или более файлов имеет вид:

```
MOVE [/Y|/-Y] [диск:][путь]имя_файла1[,...] результирующий_файл
```

Синтаксис команды для переименования папки имеет вид:

```
MOVE [/Y|/-Y] [диск:][путь]каталог1 каталог2
```

Здесь параметр *результирующий_файл* задает новое размещение файла и может включать имя диска, двоеточие, имя каталога, либо их сочетание. Если перемещается только один файл, допускается указать его новое имя. Это позволяет сразу переместить и переименовать файл. Например,

```
MOVE "C:\Мои документы\список.txt" D:\list.txt
```

Если указан ключ /-Y, то при создании каталогов и замене файлов будет выдаваться запрос на подтверждение. Ключ /Y отменяет выдачу такого запроса.

Команда *SUBST*

В Windows 9*x*, как и в MS-DOS, имеется возможность сопоставить заданному пути имя виртуального диска. Это делается с помощью команды

```
SUBST [диск1: [диск2:]путь]
```

Например, следующая команда сопоставляет пути C:\WINDOWS\SYSTEM имя диска F:

```
SUBST F: C:\WINDOWS\SYSTEM
```

Использование SUBST в этом примере позволяет быстро перейти в системный каталог, используя имя диска F:, вместо того, чтобы выбирать (или набирать на клавиатуре) длинный путь с вложенными каталогами. Сделанные с помощью SUBST сопоставления отображаются в Проводнике (Explorer) Windows (т. е. эти сопоставления действуют не только в том окне, где они сделаны) и остаются в силе до перезагрузки Windows, если только вы сами их не удалите командой

```
SUBST диск1: /D
```

Например, команда

```
C:\SUBST F: /D
```

удалит созданный ранее виртуальный диск F:. Ввод SUBST без параметров позволяет вывести текущий список виртуальных дисков.

Команды *VOL, LABEL*

С помощью команды VOL [*диск*:] можно вывести метку тома и его номер (если они существуют). Параметр *диск*: определяет логический диск, для которого нужно вывести метку и номер. Если параметр не указывается, то выводится метка и номер текущего диска.

Создать, изменить или удалить метку диска можно с помощью команды

```
LABEL [диск:][метка]
```

Кроме того, метку диска можно указать при его форматировании с помощью команды FORMAT.

Команды сравнения и поиска для файлов

Команда *FC*

Довольно часто возникает необходимость сравнить два файла (текстовых или двоичных) или два набора файлов и вывести различия между ними. Стандартной командой, предназначенной для этой цели, является команда FC.

Синтаксис для сравнения текстовых файлов имеет вид:

```
FC [/A][/C][/L][/LBn][/N][/T][/W][/nnnn] [диск1:][путь1]
  файл1 [диск2:][путь2]файл2
```

Синтаксис для сравнения двоичных файлов имеет вид:

```
FC /B [диск1:][путь1]файл1  [диск2:][путь2]файл2
```

Например, пусть имеются два текстовых файла a.txt и b.txt следующего содержания:

a.txt
А, б, в, раз, два, три, четыре, пять, шесть, семь, восемь, девять, десять

b.txt
А, б, в, раз, два, три, четыре, пять, шесть, семь, восемь, девять, десять

В результате выполнения команды

```
FC a.txt b.txt
```

на экран выведется следующая информация:

```
Сравнение файлов a.txt и b.txt
****** a.txt
А, б, в,
раз, два, три, четыре,
пять, шесть,
семь, восемь,
****** b.txt
А, б, в,
раз, два, три,
четыре, пять, шесть,
семь, восемь,
******
```

В этом примере файлы a.txt и b.txt сравнивались как текстовые файлы в формате ASCII. Явным образом этот режим сравнения задается с помощью ключа /L. При этом FC сравнивает два файла построчно и пытается вновь синхронизировать файлы после несовпадения. О найденных различиях между файлами сообщается следующим образом: выводятся имя первого файла, последняя сравниваемая строка первого файла (в нашем примере А, б, в), последняя совпавшая строка в обоих файлах (А, б, в), несовпадающие в обоих файлах строки из первого файла (раз, два, три, четыре, и пять, шесть), следующая сравниваемая строка в обоих файлах (семь, восемь). Затем выводятся имя второго файла, последняя сравниваемая строка второго файла (в нашем примере А, б, в), различающиеся строки из второго файла (раз, два, три и четыре, пять, шесть), а также следующая сравниваемая строка (семь, восемь).

Ключ /A в команде FC задает режим сокращенного вывода. В этом режиме, вместо вывода на экран всех несовпадающих строк FC выводит для каждого набора различий только первую и последнюю строки.

Ключ /C игнорирует регистр символов при сравнении.

Ключ /LB*n* задает число строк для внутреннего буфера (по умолчанию 100). Если в сравниваемых файлах содержится превышающее этот размер число последовательных различных строк, сравнение прерывается.

Ключ /N выводит номера строк при сравнении текстовых файлов ASCII. Это удобно при сравнении файлов с большим количеством строк. Например, для сравнения файлов a.txt и b.txt и вывода результата в сокращенном формате с номерами строк, нужно ввести следующую команду:

```
FC /A /N a.txt b.txt
```

При этом на экран выведется такая информация:

```
Сравнение файлов a.txt и b.txt
****** a.txt
1:  А, б, в,
...
4:  семь, восемь,
****** b.txt
1:  А, б, в,
...
4:  семь, восемь,
******
```

По умолчанию при сравнении символ табуляции интерпретируется как восемь пробелов. Использование ключа /T позволяет запретить преобразование табуляции в пробелы.

При задании /W команда FC игнорирует (и не сравнивает) пробелы в начале и в конце строки.

Ключ /nnnn задает число последовательных совпадений, после которых FC может считать синхронизацию файлов восстановленной. Если число совпадающих строк в файлах меньше этого значения, то команда FC выводит совпадающие строки как различные. Значение параметра nnnn по умолчанию равно 2.

Ключ /B позволяет сравнивать файлы в двоичном режиме. При этом FC сравнивает два файла побайтно и не пытается заново синхронизировать их после несовпадения. Для вывода отчета о несовпадениях при двоичном сравнении используется следующий формат:

```
xxxxxxxx: yy zz
```

где значение xxxxxxxx задает относительный шестнадцатеричный адрес для пары байтов, считая от начала файла. Адреса начинаются с 00000000. Шестнадцатеричные значения yy и zz представляют несовпавшие байты из первого и второго файлов.

В любом из имен сравниваемых файлов можно использовать групповые символы * и ?. При указании такого символа в имени первого файла команда FC сравнивает все заданные файлы со вторым файлом. Если же указать групповой символ в имени второго файла, FC использует соответствующее значение из имени первого файла.

Команда *FIND*

Еще одной распространенной задачей является поиск заданной строки текста в одном или нескольких файлах. Поиск с помощью стандартных средств Windows (**Пуск/Поиск/Файлы и папки** (Start/Find/Files and folders)) позволяет получить список всех таких файлов, однако здесь не видно, в каком именно месте файла содержится строка, которую мы искали. С помощью команды FIND можно вывести на экран (перенаправить в файл или на принтер) все строки текста из одного или нескольких файлов, содержащих (или, наоборот, не содержащих) заданную строку; при этом имеется возможность вывода номеров найденных строк.

Синтаксис команды FIND имеет вид:

```
FIND [/V][/C][/N][/I] "строка" [[диск:][путь]имя_файла[ ...]]
```

Параметр *"строка"* (указывается в кавычках) задает группу символов, которую вы хотите найти. Если эта строка содержит кавычки, для каждого вхождения кавычек внутри строки нужно указывать символ кавычки дважды. Команда FIND не распознает возвратов каретки, поэтому для поиска в файле текста, включающего возврат каретки, нужно ограничить строку текстом до возврата каретки.

Параметр [*диск*:][*путь*]*имя_файла* задает расположение и имя файла, в котором будет происходить поиск. Можно указывать через пробел несколько

файлов для поиска. Например, для того чтобы вывести все строки из файлов music1.dat и music2.dat, содержащие строку "Pink Floyd", нужно задать команду

```
FIND "Pink Floyd" music1.dat music2.dat
```

Если путь для поиска не задан, команда FIND производит поиск в тексте, введенном с клавиатуры, либо переданном по конвейеру другой командой (процесс конвейеризации команд будет подробно описан ниже).

Ключ /V выводит все строки из файлов, используемых для поиска, не содержащие заданную строку. То есть, для того чтобы вывести все строки из файлов music1.dat и music2.dat, *не* содержащие строку "Pink Floyd", нужно задать команду

```
FIND /V "Pink Floyd" music1.dat music2.dat
```

Ключ /C выводит только число строк, содержащих заданную строку, /N выводит перед каждой строкой номер строки, /I задает игнорирование регистра символов при поиске.

После завершения своей работы команда FIND возвращает коды, описанные в табл. 1.5.

Таблица 1.5. *Коды завершения команды* FIND

Код	Описание
0	Поиск завершен успешно. Найдено, по крайней мере, одно совпадение
1	Поиск завершен успешно, но совпадений не найдено
2	При поиске произошла ошибка, выполнение команды прервано. В этом случае нельзя ничего сказать о том, были ли найдены совпадения до возникновения ошибки

Команда *FOR*

Отметим, что в имени файла или в расширении, задаваемых в команде FIND, нельзя использовать групповые символы * и ?. Для поиска строки в наборе файлов можно воспользоваться командой FOR.

Вообще, с помощью FOR выполняется запуск определенной команды для каждого файла или набора файлов. Команду FOR можно использовать как в командной строке, так и в пакетных файлах, причем синтаксис для этих случаев немного различается.

Синтаксис для командной строки имеет вид:

```
FOR %переменная IN (набор) DO команда [параметры]
```

Внимание!

Отметим, что IN и DO — это не параметры, а обязательные ключевые слова команды FOR

Параметр *%переменная* представляет собой подставляемую переменную. Команда FOR заменяет эту переменную текстом каждой строки в заданном множестве, пока *команда* после ключевого слова DO не обработает все файлы.

Параметр (*набор*) включает в себя один или более файлов (скобки обязательны), причем можно использовать групповые символы. Допустимы, например, следующие множества: (*.doc), (*.doc *.dat *.rpt), (ol??2001.* kr??2001.*).

При использовании команды FOR первое значение в указанном наборе заменяет параметр *%переменная*, и Windows выполняет для обработки этого значения заданную команду. Этот процесс продолжается, пока не обработаются все файлы (или группы файлов) в наборе.

Например, для распечатки (с помощью команды PRINT) содержимого всех файлов с расширением dat и rpt в текущем каталоге можно воспользоваться следующей командой:

```
FOR %f IN (*.dat *.rpt) DO PRINT %f
```

Возвращаясь к задаче поиска строки в группе файлов, приведем следующий пример: выведем строки из всех файлов в текущем каталоге, название которых начинается с music, и которые содержат строку "Pink Floyd". Для этого можно использовать следующую команду:

```
FOR %f IN (music*.*) DO FIND "Pink Floyd" %f
```

Замечание

В Windows 9*x*, как и в Windows NT, нет стандартной команды, с помощью которой можно было бы осуществлять поиск строки в файле с одновременной заменой этой строки на другую. Такую возможность предоставляет утилита командной строки MUNGE из пакета Resource Kit для Windows NT *(см. разд. "Поиск и замена текста в файлах" гл. 3)*.

Перенаправление ввода/вывода и конвейеризация команд

Многие команды Windows используют стандартные средства ввода и вывода информации: данные читаются со стандартного входного устройства (обычно это клавиатура), сообщения выводятся на стандартное выходное устройство (как правило, это монитор). Рассмотрим команды, которые позволяют изменить стандартное устройство ввода/вывода.

Команда *CTTY* и перенаправление ввода/вывода

С помощью команды CTTY можно сменить устройство, используемое системой в качестве терминала (стандартного устройства ввода/вывода). Синтаксис этой команды имеет вид:

```
CTTY терминал
```

Здесь параметр *терминал* задает альтернативное устройство, которое будет использоваться для набора команд. Допустимыми значениями этого параметра являются PRN, LPT1, LPT2, LPT3, CON, AUX, COM1, COM2, COM3 и COM4. Например, чтобы переопределить весь ввод и вывод с текущего устройства (монитора и клавиатуры компьютера) на порт AUX, нужно выполнить следующую команду:

```
CTTY AUX
```

Кроме команды CTTY для задания устройства ввода можно также использовать параметр в команде COMMAND *(см. ранее разд. "Командный интерпретатор command.com" данной главы).*

Кроме этого, в Windows поддерживается UNIX-подобная концепция переназначения устройств стандартного ввода и стандартного вывода. Эта концепция определяет способ, с помощью которого одна программа направляет свой вывод на вход другой или перехватывает вывод другой программы в качестве своих входных данных. Таким образом, имеется возможность передавать информацию от процесса к процессу при минимальных программных издержках. Практически это означает, что для программ, которые используют стандартные входные и выходные устройства, операционная система позволяет:

- выводить сообщения программ не на экран, а в файл или на принтер (перенаправление вывода);
- читать входные данные не с клавиатуры, а из заранее подготовленного файла (перенаправление ввода);
- передавать сообщения, выводимые одной программой, в качестве входных данных для другой программы (конвейеризация команд).

Из командной строки эти возможности реализуются следующим образом. Для того чтобы перенаправить сообщения, выводимые какой-либо командой, в текстовый файл, нужно использовать конструкцию

```
команда > имя_файла
```

Если при этом заданный для вывода файл уже существовал, то он перезаписывается (старое содержимое теряется), если не существовал — создается. Можно также не создавать файл заново, а дописывать информацию, выво-

димую командой, в конец существующего файла. Для этого команда перенаправления вывода должна быть задана так:

команда >> *имя_файла*

С помощью символа < можно прочитать входные данные для заданной команды не с клавиатуры, а из определенного (заранее подготовленного) файла:

команда < *имя_файла*

Наконец, с помощью конструкции

команда1 | *команда2*

можно использовать сообщения, выводимые первой командой, в качестве входных данных для второй команды.

Приведем несколько примеров.

1. Вывод встроенной справки для команды COPY в файл copy.txt:

   ```
   COPY /? > copy.txt
   ```

2. Добавление текста справки для команды XCOPY в файл copy.txt:

   ```
   XCOPY /? >> copy.txt
   ```

3. Ввод новой даты из файла date.txt (DATE — это команда для просмотра и изменения системной даты):

   ```
   DATE < date.txt
   ```

Команды *MORE* и *SORT*

Одной из наиболее часто использующихся команд, для работы с которой применяется перенаправление ввода/вывода и конвейеризация, является MORE. Эта команда считывает стандартный ввод из конвейера или перенаправленного файла и выводит информацию частями, размер каждой из которых не больше размера экрана. Используется MORE обычно для просмотра длинных файлов. Возможны три варианта синтаксиса этой команды:

MORE [*диск*:][*путь*]*имя_файла*
MORE < [*диск*:][*путь*]*имя_файла*
имя_команды | MORE

Параметр [*диск*:][*путь*]*имя_файла* определяет расположение и имя файла с просматриваемыми на экране данными. Параметр *имя_команды* задает команду, вывод которой отображается на экране (например, DIR или команда TYPE, использующаяся для вывода содержимого текстового файла на экран).

Приведем два примера.

1. Для поэкранного просмотра помощи команды XCOPY используется команда:

```
XCOPY /? | MORE
```

2. Для поэкранного просмотра текстового файла news.txt возможны следующие варианты команд:

```
MORE news.txt
MORE < news.txt
TYPE news.txt | MORE
```

Другой распространенной командой, использующей перенаправление ввода/вывода и конвейеризацию, является SORT. Эта команда работает как фильтр: она считывает символы в заданном столбце, упорядочивает их в возрастающем или убывающем порядке и выводит отсортированную информацию в файл, на экран или другое устройство. Возможны два варианта синтаксиса этой команды:

```
SORT [/R] [/+n] [[диск1:][путь1]файл1] [> [диск2:][путь2]файл2]
```

или

```
[команда |] SORT [/R] [/+n] [> [диск2:][путь2]файл2]
```

В первом случае параметр *[диск1:][путь1]файл1* определяет имя файла, который нужно отсортировать. Во втором случае будут отсортированы выходные данные указанной команды. Если параметры *файл1* или *команда* не заданы, то SORT будет считывать данные с устройства стандартного ввода.

Параметр *[диск2:][путь2]файл2* задает файл, в который будет направляться сортированный вывод; если этот параметр не задан, то вывод будет направлен на устройство стандартного вывода.

Замечание

В команде SORT нельзя использовать групповые символы в именах файлов. Если необходимо сортировать группу файлов, то нужно совместно использовать SORT и FOR.

По умолчанию сортировка выполняется в порядке возрастания. Ключ /R позволяет изменить порядок сортировки на обратный (от Z к A и затем от 9 до 0). Например, для поэкранного просмотра отсортированного в обратном порядке файла price.txt, нужно задать следующую команду:

```
SORT /R < price.txt |MORE
```

Ключ /+n задает сортировку в файле по символам n-го столбца. Например, /+10 означает, что сортировка должна осуществляться, начиная с 10-й позиции в каждой строке. По умолчанию файл сортируется по первому столбцу.

Команда *ECHO*

Используя механизм перенаправления ввода/вывода, можно из командной строки посылать информацию на различные устройства и автоматизировать ответы на запросы, выдаваемые командами или программами, использующими стандартный ввод. Для решения таких задач подходит команда

```
ECHO [сообщение]
```

которая выводит сообщение на экран. Рассмотрим примеры использования этой команды.

1. Посылка символа прогона на принтер:

   ```
   ECHO ^L > PRN
   ```

2. Удаление всех файлов в текущем каталоге без предупреждения (автоматический положительный ответ на запрос об удалении):

   ```
   ECHO y | DEL *.*
   ```

3. Соединение по телефону из командной строки (модем связан с портом COM2):

   ```
   ECHO ATDT 1(123)555-1234 > COM2
   ```

Работа с переменными среды

После загрузки Windows 9*x* в оперативной памяти постоянно хранится набор так называемых *переменных среды* (или *переменных окружения*). Каждая переменная среды имеет свое уникальное имя, а значением такой переменной всегда является строка. В MS-DOS переменные среды часто использовались для хранения и передачи информации, необходимой для функционирования одной или нескольких задач. Например, в переменной мог быть указан путь к каталогу, в котором прикладные программы должны искать требующиеся им файлы. Для того чтобы прикладная задача могла получить значение какой-либо переменной, достаточно было знать только ее имя. В Windows, как вы знаете, существует более совершенный механизм хранения информации, необходимой для приложений, — это системный реестр, в котором, в конечном счете, также записаны имена переменных и их значения. "Правильная" Windows-программа должна использовать именно реестр, а не переменные среды. Однако если вы пользуетесь старыми приложениями, то от нужных им переменных среды отказаться не удастся.

Кроме этого, переменные среды необходимы самой операционной системе. Некоторые переменные устанавливаются автоматически при запуске Windows, исходя из содержания конфигурационного файла `msdos.sys` в корневом каталоге загрузочного диска. Такими переменными являются, например, `WINDIR`, которая определяет расположение каталога Windows (`WINDIR=C:\WINDOWS`),

или TEMP, которая определяет каталог для хранения временных файлов Windows (TEMP=C:\WINDOWS\TEMP). Другие переменные инициализируются также при загрузке системы с помощью специальной команды SET, помещенной в файл autoexec.bat. Наиболее важной переменной такого типа является PATH, в которой хранится *системный путь* (путь поиска), т. е. список каталогов, в которых система должна искать выполняемые файлы или файлы совместного доступа (например, динамические библиотеки). Если имя каталога указано в системном пути, то можно запускать программу, находящуюся в этом каталоге, не указывая путь к ней. В частности, каталог, в котором хранятся исполняемые файлы, представляющие стандартные внешние команды Windows (например, XCOPY), всегда включается в системный путь, поэтому в командной строке мы используем только имя внешней команды, не задумываясь о ее местонахождении. Каждый каталог, входящий в переменную PATH, должен отделяться от другого точкой с запятой. При этом длинные имена должны сокращаться до их коротких аналогов, т. е. значение системного пути выглядит примерно так:

```
PATH=C:\WINDOWS;C:\WINDOWS\COMAND;C:\PROGRA~1
```

Замечание

Каталоги \WINDOWS, \WINDOWS\COMMAND добавляются в переменную PATH операционной системой автоматически.

В файле autoexec.bat также часто устанавливается переменная PROMPT, которая определяет вид приглашения в командной строке. Приглашение может включать обычные символы и специальные коды, представленные в табл. 1.6.

Таблица 1.6. *Коды, использующиеся в переменной PROMPT*

Код	Символ	Код	Символ
$Q	= (знак равенства)	$$	$ (символ доллара)
$T	Текущее время	$D	Текущая дата
$P	Текущие диск и путь	$V	Номер версии DOS или Windows
$N	Текущий диск	$G	> (знак "больше")
$L	< (знак "меньше")	$B	| (вертикальная черта)
$H	Backspace (удаление предыдущего символа)	$E	Код Escape (ASCII 27)
$_	Возврат каретки и перевод строки		

Команда *SET*

Вывести на экран, установить значения и удалить переменные среды можно с помощью команды

SET [переменная=[строка]]

Отметим сразу, что изменения, сделанные в переменных среды с помощью команды SET, имеют силу только для текущего командного окна. Например, вы можете в командном окне изменить значение переменной PATH, однако в другом командном окне значение PATH останется старым. Сделать "глобальное" изменение в переменной можно только внеся нужные поправки в autoexec.bat и перезагрузив компьютер.

Запуск команды SET без параметров позволяет вывести текущие значения переменных среды. Если в команде задаются имя переменной и непустая строка (в кавычки ее заключать не нужно), то система добавляет заданную переменную к операционной среде текущего командного окна и связывает строку с этой переменной. Например, команда

```
SET MyVar=Hello!
```

определяет новую переменную MyVar со значением "Hello!". Если переменная уже существует, то новое значение строки заменяет старое. То есть, введя команду

```
SET MyVar=Bye!
```

мы изменим значение переменной MyVar на "Bye!". Если вы задаете только переменную и знак равенства (без строки), то система сбрасывает связанное с переменной значение строки. Поэтому, введя команду

```
SET MyVar=
```

мы удалим переменную MyVar.

Замечание

Для работы с переменными PATH и PROMPT, кроме команды SET используются специальные одноименные команды: PATH [[*диск*:]*путь*[;...]] и PROMPT [*текст*].

Значения переменных среды (текстовые строки) можно использовать в командных файлах и при работе с командной строкой. Для получения значения переменной ее имя нужно заключить в знаки %.

Рассмотрим примеры использования переменных среды.

1. Вывод списка файлов, содержащихся в каталоге для хранения временных файлов, путь к которому сохранен в переменной TEMP:

```
DIR %TEMP%
```

2. Добавление пути D:\MyPrograms к переменной PATH:

```
SET PATH=D:\MyPrograms\;%PATH%
```

Запуск внешних программ и документов

Любой выполняемый файл (с расширением com или exe) в Windows можно запустить из командной строки, просто набрав его имя (заключенное, при необходимости, в двойные кавычки) и нажав клавишу <Enter>. Кроме того, существует специальная команда START, с помощью которой можно:

- запускать программы в определенном режиме;
- открывать окно просмотра для заданного каталога (папки);
- открывать документы, тип которых зарегистрирован в Windows (т. е. указано, какое приложение должно открывать документ с заданным расширением).

Синтаксис команды для запуска программ:

START [*ключи*] *программа* [*аргументы...*]

Синтаксис команды для открытия окна просмотра:

START [*ключи*] *каталог*

Синтаксис команды для открытия документов:

START [*ключи*] *документ.расш*

Возможные значения ключей описаны в табл. 1.7.

Таблица 1.7. *Ключи команды* START

Параметр	Описание
/M[inimized]	Запуск программы в свернутом окне (фоновый режим)
/MAX[imized]	Запуск программы в развернутом окне (основной процесс)
/R[estored]	Запуск программы в стандартном окне (используется по умолчанию)
/W[ait]	Отложить запуск до завершения предыдущей программы

Приведем несколько примеров использования команды START.

1. Запуск из командной строки в качестве основного процесса Microsoft Word с автоматическим открытием двух заданных файлов (имя одного из файлов содержит пробел, поэтому оно взято в кавычки):

```
START /MAX winword.exe "Мой документ.doc" docum.doc
```

2. Открытие окна просмотра для каталога Мои документы на текущем диске:

```
START "\Мои документы"
```

3. Открытие окна просмотра для текущего каталога:

```
START .
```

4. Открытие файла Мой любимый документ.doc из папки Мои документы в стандартном окне:

```
START "C:\Мои документы\Мой любимый документ.doc"
```

Команды для работы с локальной сетью

При работе с локальной сетью наиболее часто решаются следующие задачи: организация общих сетевых ресурсов, подключение пользователей к этим ресурсам, организация резервного копирования данных с одного компьютера на другой. Напомним, что для доступа к сетевым ресурсам (предоставленным в общее пользование файлам или принтерам) используются *сетевые пути* в следующем формате:

```
\\Имя_компьютера\Имя_ресурса[\Имя_папки\...\Имя_файла]
```

Например, если на компьютере с сетевым именем Server1 имеется общедоступный принтер с сетевым именем Epson, то путь к нему указывается так: \\Server1\Epson. Если на этом же компьютере имеется общедоступная папка (каталог) Programs, то путь к ней выглядит как \\Server1\Programs, а путь к отдельному файлу в этой папке, например, так:

```
\\Server1\Programs\Word97\winword.exe
```

В Windows многие команды, применяемые для работы с файлами, могут обрабатывать и сетевые пути. Например, допустима следующая команда:

```
COPY \\Server1\Programs\Word97\winword.exe \\Server2\Programs
```

Такую возможность имеют почти все из рассмотренных выше команд Windows: ATTRIB, COPY, XCOPY, DIR, MKDIR, RMDIR, DELTREE, REN, MOVE, FC, SORT, FIND, FOR, START. Команды CD и SUBST не могут работать с сетевыми путями. Кроме этого, сетевой маршрут можно использовать при операциях перенаправления ввода/вывода, например

```
DIR > \\Server1\Programs\list.txt
```

Команда *NET*

Многие задачи, связанные с использованием локальной сети и сетевых ресурсов, в Windows можно решить с помощью специальной команды NET, возможные варианты которой приведены в табл. 1.8.

Таблица 1.8. *Основные сетевые команды Windows 9x (NET ...)*

Команда	Описание
NET CONFIG	Вывод сведений о рабочей группе
NET DIAG	Запуск программы Microsoft Network Diagnostics для получения данных о сети
NET HELP	Вывод сведений о командах и сообщений об ошибках
NET INIT	Загрузка протокола и драйверов сетевой платы без привязки их к диспетчеру протоколов
NET LOGOFF	Отключение всех используемых компьютером общих ресурсов
NET LOGON	Идентификация пользователя как члена рабочей группы
NET PASSWORD	Изменение пароля для входа в сеть
NET PRINT	Вывод сведений об очередях печати и управление заданиями по выводу на печать
NET START	Запуск служб
NET STOP	Остановка работы служб
NET TIME	Вывод времени с другого компьютера или синхронизация часов с часами на сервере времени Microsoft Windows или Novell NetWare
NET USE	Подключение и отключение сетевых ресурсов и вывод сведений об имеющихся подключениях
NET VER	Вывод типа и версии используемой системы переадресации
NET VIEW	Вывод списка компьютеров, обеспечивающих совместный доступ к ресурсам, или общих ресурсов конкретного компьютера

Отметим, что в Windows 9x нельзя из командной строки предоставить папку или принтер на своем компьютере в общее пользование (в Windows NT такая возможность существует) — сделать это можно только с помощью Проводника (Explorer) Windows. Для этого нужно: выделить папку в Проводнике (Explorer), щелкнуть правой кнопкой мыши и выбрать пункт **Доступ** (Sharing). В открывшемся диалоговом окне (рис. 1.6) нужно указать сетевое имя создаваемого ресурса, тип доступа к нему и, в случае необходимости, пароли на чтение и запись.

Рассмотрим теперь более подробно наиболее часто используемые команды из табл. 1.8: NET VIEW, NET USE, NET PRINT, NET TIME.

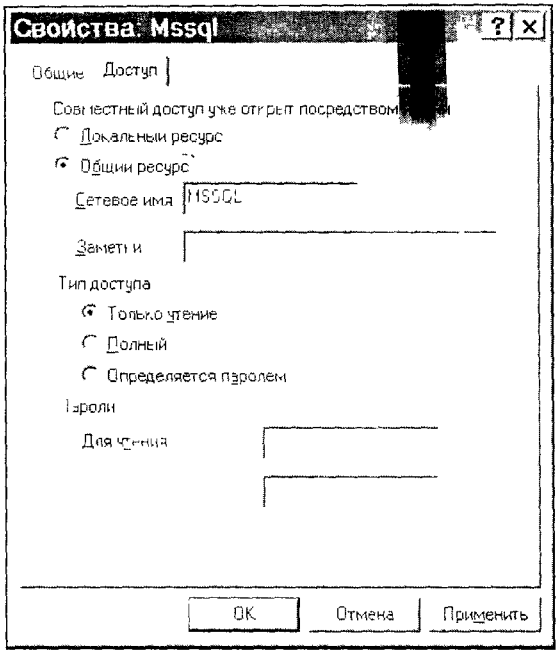


Рис. 1.6. Создание общего ресурса в Windows 95

Команда *NET VIEW*

Начнем с команды `NET VIEW`. Возможны два варианта синтаксиса этой команды:

```
NET VIEW [\\компьютер] [/YES]
NET VIEW [/WORKGROUP:группа] [/YES]
```

Используемые параметры и ключи описаны в табл. 1.9.

Таблица 1.9. *Параметры и ключи команды* `NET VIEW`

Параметр	Описание
компьютер	Имя компьютера, список общих ресурсов которого следует вывести
`/WORKGROUP`	Этот ключ указывает необходимость вывода списка компьютеров из другой рабочей группы, имеющих общие ресурсы
группа	Имя рабочей группы, список компьютеров которой следует вывести
`/YES`	Выполнение команды `NET VIEW` без предварительного запроса данных или подтверждения

Например, для запроса информации о доступных ресурсах на компьютере Server1 можно воспользоваться командой

```
NET VIEW \\Server1
```

Ответ может быть примерно таким:

```
Общие ресурсы компьютера \\Server1
Сетевое имя  Тип          Заметки
-----------------------------------------------------------------
PROGRAMS     Диск         Общие программы
HP           Печать
```

Для вывода полного списка компьютеров, имеющих общие ресурсы, из рабочей группы, в которой зарегистрирован данный компьютер, используется команда NET VIEW без параметров. В результате ее выполнения выводится информация следующего типа:

```
Серверы, доступные рабочей группе WORKGROUP.
Сервер                Заметки
-----------------------------------------------------------------
\\Server1             Основной сервер
\\Server2             Резервный сервер
\\Comp1               Рабочая станция
```

Команда *NET USE*

Перейдем к команде NET USE. Для того чтобы из командной строки подключить общую папку в качестве сетевого диска, используется следующий вариант этой команды:

```
NET USE [диск:|*] [\\компьютер\папка [пароль|?]] [/SAVEPW:NO][/YES][/NO]
```

Параметр *диск*: — это имя диска, назначаемое общей папке. Например, для подключения диска H: к сетевому ресурсу Programs на сервере Server1 используется следующая команда:

```
NET USE H: \\Server1\Programs
```

Можно явно и не указывать имя подключаемого диска, указав параметр *. В этом случае система присвоит новому сетевому диску первую свободную букву.

В случае необходимости в команде NET USE можно указать пароль к запрашиваемому сетевому ресурсу. Если указать параметр ?, то пароль для доступа к ресурсу будет запрашиваться интерактивно. В том случае, когда ввод пароля необязателен, данный параметр можно не указывать. Использование ключа /SAVEPW:NO позволяет предотвратить запись пароля в файл со списком паролей. В этом случае при каждом подключении к ресурсу пароль надо будет вводить заново.

Иногда при использовании команды NET USE возникает необходимость ответить на тот или иной вопрос (например, если имя диска, которое вы пытаетесь присвоить, уже занято другим сетевым диском). Ключи /YES и /NO позволяют автоматизировать ответы на такие вопросы: /YES приводит к выполнению команды без предварительного запроса данных или подтверждения, /NO — к выполнению команды с автоматической выдачей отрицательных ответов на все запросы, относящиеся к подтверждению действий.

Если вы зарегистрированы в домене Windows NT и у вас имеется отмеченный в регистрационной записи домашний каталог, то подключить сетевой диск к такому каталогу можно с помощью команды

```
NET USE диск:|* /HOME
```

Подключить локальный порт компьютера к сетевому принтеру можно с помощью команды

```
NET USE [порт:] [\\компьютер\принтер [пароль|?]] [/SAVEPW:NO][/YES][/NO]
```

Параметры и ключи здесь имеют то же значение, что и в команде NET USE для подключения сетевого диска. Например, для подключения первого параллельного порта к сетевому принтеру Epson на сервере Server1 используется следующая команда:

```
NET USE LPT1: \\Server1\Epson
```

Команда NET USE без параметров выводит список всех подключенных ресурсов (сетевых дисков и принтеров).

Отключить один сетевой диск можно с помощью следующей команды:

```
NET USE диск:|\\компьютер\папка /DELETE [/YES]
```

Например, для отключения подключенного ранее сетевого диска H: используется следующая команда:

```
NET USE H: /DELETE
```

Аналогичная команда применяется для отмены связи локального порта с сетевым принтером:

```
NET USE порт:|\\компьютер\принтер /DELETE [/YES]
```

Например:

```
NET USE LPT1: /DELETE.
```

Команда

```
NET USE * /DELETE [/YES]
```

отключает сразу все подключенные сетевые ресурсы.

Команда *NET PRINT*

Перейдем теперь к команде NET PRINT. Получить информацию об очередях печати на сетевом ресурсе можно с помощью следующего варианта этой команды:

```
NET PRINT \\компьютер[\принтер]|порт [/YES]
```

Параметр *порт* здесь означает имя параллельного порта (LPT), назначенного интересующему нас принтеру. Например, если порт LPT1 переназначен на сетевой принтер \\Server1\Epson, то можно использовать команду

```
NET PRINT \\Server1\Epson
```

или

```
NET PRINT LPT1:
```

Если в команде NET PRINT задано только имя компьютера, то на экран выводятся сведения о всех очередях печати общих принтеров, присоединенных к этому компьютеру.

Управлять заданиями по выводу на печать можно с помощью команды

```
NET PRINT \\компьютер|порт [задание# [/PAUSE|/RESUME|/DELETE]] [/YES]
```

Параметр *задание#* соответствует номеру, который присвоен заданию, поставленному в очередь по выводу на печать. Ключ /PAUSE позволяет приостановить это задание, ключ /RESUME — продолжить вывод остановленного задания, ключ /DELETE — удалить задание.

Команда *NET TIME*

Последней командой, которую мы рассмотрим в этом разделе, будет NET TIME. Эта команда предназначена для просмотра системного времени на другом компьютере и, в случае необходимости, синхронизации времени на своем компьютере с часами этого компьютера. Синтаксис этой команды:

```
NET TIME [\\компьютер|/WORKGROUP:группа] [/SET] [/YES]
```

Например, для того чтобы посмотреть текущее время на компьютере \\Server1, нужно использовать команду

```
NET TIME \\Server1
```

Ключ /WORKGROUP:*группа* указывает на необходимость использования часов компьютера из другой рабочей группы. Если в указанной рабочей группе есть выделенный сервер времени, то команда

```
NET TIME /WORKGROUP:группа
```

выведет системное время этого сервера.

С помощью ключа /SET можно синхронизировать часы локального компьютера с часами указанного компьютера или рабочей группы. Например:

```
NET TIME \\Server1 /SET
```

Ключ /YES означает, как обычно, что команда NET TIME будет выполняться без предварительного запроса данных или подтверждения.

Команды конфигурации системы

Напомним вначале схему загрузки Windows 9*x* после включения компьютера. Эти операционные системы основаны на MS-DOS, поэтому вначале происходит загрузка базовых системных файлов MS-DOS с жесткого диска. Первым в это время обрабатывается двоичный файл io.sys (редактировать его нельзя), который запускает необходимые драйверы (himem.sys, ifshlp.sys) и устанавливает некоторые параметры системы. В MS-DOS и Windows 3.x эти параметры задавались с помощью специальных команд в файле config.sys; в Windows 9*x* можно, пользуясь теми же командами в config.sys, изменить установленные по умолчанию значения этих параметров. Файл io.sys при своей загрузке выполняет следующие команды конфигурации системы.

- BUFFERS=x. Эта команда определяет число файловых буферов для уменьшения времени доступа к диску тех приложений, которые используют для работы с файлами вызовы функций MS-DOS. Значение по умолчанию, устанавливаемое в io.sys, равно 30; в config.sys можно увеличить это значение.
- DOS=HIGH. В случае необходимости можно изменить эту настройку, указав в config.sys команду DOS=LOW.
- FCBS=x. Создает заданное число блоков управления файлами (FCB, File Control Block) для старых приложений MS-DOS. Значение по умолчанию равно 4; в случае необходимости можно увеличить это значение в config.sys.
- FILES=x. Значение по умолчанию равно 60. Эта команда устанавливает число файловых буферов для работы приложений MS-DOS с открытыми файлами; для приложений Win16 или Win32 этот параметр не требуется. В config.sys можно установить другое значение, которое должно быть больше значения по умолчанию.
- LASTDRIVE=x. Эта команда устанавливает последнюю букву в алфавите, которая может использоваться для обозначения имени диска. По умолчанию устанавливается буква Z.
- SHELL=COMMAND.COM. Устанавливается в io.sys по умолчанию, в случае необходимости в config.sys можно установить другое значение параметра SHELL.

- STACKS=x. Устанавливает число и размер стеков для использования старыми приложениями. Значение по умолчанию равно 9,256, т. е. определяется 9 стеков по 256 байтов каждый. В config.sys можно установить другие значения для этого параметра.

Далее при загрузке считывается текстовый файл msdos.sys, в котором записаны параметры для инициализации Windows. Эти параметры и их возможные значения представлены в табл. 1.10.

Таблица 1.10. *Возможные установки в* msdos.sys

Параметр	Описание
HostWinBootDrv=x	Определяет букву диска, с которого загружается Windows
WinBootDir=x	Определяет каталог, в который установлена Windows. Обычно это C:\WINDOWS
WinDir=x	Определяет каталог, в который установлена Windows. Обычно указывается то же значение, что и для WinBootDir
BootDelay=x	Указывает временной интервал, в течение которого отображается заставка "Loading Windows 95/98"; в течение этого промежутка времени пользователь может, нажав клавишу <F8>, попасть в загрузочное меню Windows
BootFailSafe=x	Если значение равно 1, то система загружается в режиме защиты от сбоев. По умолчанию устанавливается в 0 (загрузка в обычном режиме)
BootGUI=x	При значении, равном 1 (по умолчанию), происходит загрузка графического пользовательского интерфейса Windows 98. Если значение равно 0, то происходит загрузка в режиме MS-DOS
BootKeys=x	Если значение равно 1 (по умолчанию), то для выбора способа загрузки можно пользоваться функциональными клавишами (<F5>, <F6> и <F8>). При значении, равном 0, функциональные клавиши недоступны
BootMenu=x	При значении 1 автоматически отображается загрузочное меню Windows. Значение, равное 0, говорит о том, что загрузка проводится обычным способом
BootMenuDefault=x	Этот параметр задает пункт загрузочного меню, использующегося по умолчанию (если пользователь не сделал свой выбор в течение времени, определенного параметром BootMenuDelay)
BootMenuDelay=x	Устанавливает продолжительность (в секундах) паузы загрузочного меню. По умолчанию равно 30 секундам

Таблица 1.10 (окончани

Параметр	Описание
BootMulti=x	Значение 1 (по умолчанию) позволяет выбор при загрузк между прежней версией MS-DOS и Windows. Значение делает невозможным такой выбор
BootWarn=x	По умолчанию равно 1, что приводит к появлению пункт **Режим защиты от сбоев** (Safe mode) в загрузочном менк При значении, равном 0, происходит отключение предупре ждения **Режим защиты от сбоев** (Safe mode), равно как всего загрузочного меню
BootWin=x	Если значение равно 1 (по умолчанию), происходит загрузк Windows. При значении, равном 0, Windows не используетс в качестве операционной системы по умолчанию, что позвс ляет загружать более ранние версии MS-DOS (если он имеются)
DblSpace=x	Значение 1 (по умолчанию) позволяет поддерживать диски уплотненные с помощью программы DoubleSpace. При зна чении, равном 0, такие диски становятся недоступными
DoubleBuffer=x	Установка значения этого параметра равным 1 позволяе двойную буферизацию при работе с любыми подключенны ми устройствами SCSI. По умолчанию равно 0, что приводи к отключению режима двойной буферизации для таких уст ройств
DrvSpace=x	Значение 1 (по умолчанию) позволяет поддерживать диски уплотненные с помощью программы DriveSpace. При значе нии, равном 0, такие диски становятся недоступными
LoadTop=x	Если значение равно 1, то файлы `command.com` и `drvspace.bin` загружаются в верхнюю часть оперативной па мяти реального режима; это может потребоваться для со вместимости с некоторыми драйверами реального времени По умолчанию значение этого параметра равно 0, при этом файлы `command.com` и `drvspace.bin` загружаются в любое свободное место оперативной памяти реального режима
Logo=x	При значении, равном 1 (по умолчанию), при загрузке Windows выводится анимированная заставка. Если значение равно 0, то заставка не выводится
Network=x	Значение, равное 1, говорит о том, что в загрузочное меню Windows следует внести пункт **Режим защиты от сбоев с поддержкой сети** (Safe mode with network support). При значении, равном 0, этот пункт в загрузочное меню не вносится. Если в системе установлена поддержка сети, то значением по умолчанию является 1, иначе — 0

Затем выполняются все команды из файла config.sys (если он есть). Как отмечалось выше, в этом файле можно переопределить значения параметров, установленных по умолчанию в файле io.sys, и загрузить драйверы реального времени. Такие драйверы требуются, например, для выбора кодовой страницы (переключения на русский язык) при работе с приложениями MS-DOS или командной строкой. Для этой цели в файл config.sys включаются строки вида:

```
Device=C:\WINDOWS\COMMAND\display.sys con=(ega,,1)
Country=007,866,C:\WINDOWS\COMMAND\country.sys
```

После config.sys выполняется файл autoexec.bat (если он есть). Так как autoexec.bat является обычным командным файлом, в нем можно выполнять все описанные во второй главе команды. Обычно в этом файле устанавливаются переменные среды, подготавливается и загружается нужная для режима MS-DOS кодовая страница и драйвер-русификатор. Например:

```
PATH C:\WINDOWS;C:\WINDOWS\COMMAND;C:\NC;C:\TOOLS
mode con codepage prepare=((866) C:\WINDOWS\COMMAND\ega3.cpi) >nul
mode con codepage select=866 >nul
keyb ru,C:\WINDOWS\COMMAND\keybrd3.sys >nul
SET CLIPPER=f80
```

Отметим еще раз, что в Windows 9*x* следующие установки выполняются автоматически файлом io.sys (в autoexec.bat вы можете при необходимости их изменить):

```
SET PATH=C:\WINDOWS;C:\WINDOWS\COMMAND
PROMPT=$p$g
SET TMP=C:\WINDOWS\TEMP
SET TEMP=C:\WINDOWS\TEMP
SET COMPSPEC=C:\WINDOWS\COMMAND.COM
```

После того как файл autoexec.bat выполнен, происходит непосредственная загрузка графической среды Windows.

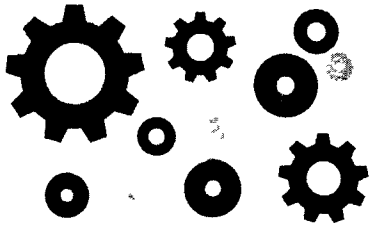
Заключение

Мы рассмотрели наиболее часто используемые команды, знание которых просто необходимо для грамотной работы в операционной системе Windows. К сожалению, в Windows 9*x* имеется довольно ограниченный (по сравнению, например, с операционной системой UNIX) набор стандартных команд и утилит командной строки, с помощью которых иногда бывает невозможно выполнить ту или иную задачу средствами только командной строки. Кроме того, как уже отмечалось выше, в Windows 9*x* не вошли неко-

торые весьма полезные команды MS-DOS. Эти недостатки в значительной степени исправлены в пакете Windows NT Resource Kit.

Ясно, что для удобства работы необходим механизм, позволяющий выполнять одни и те же команды (может быть, с небольшими изменениями) для реализации каких-то периодически повторяемых действий. Эта возможность реализуется с помощью рассматриваемых во второй главе командных файлов, которые появились еще в самых ранних версиях MS-DOS.

ГЛАВА 2



Командные файлы в Windows 9*x*

Командный (или *пакетный*) файл в Windows 9*x* — это обычный текстовый файл с расширением `bat`, в котором записаны допустимые команды операционной системы, а также некоторые дополнительные инструкции и ключевые слова, придающие командным файлам некоторое сходство с алгоритмическими языками программирования. Например, если записать в файл `deltmp.bat` следующие команды:

```
C:\
CD %TEMP%
ATTRIB -R *.tmp
DEL *.tmp
```

и запустить его на выполнение, то мы удалим все файлы во временном каталоге Windows. Таким образом, исполнение командного файла приводит к тому же результату, что и последовательный ввод записанных в нем команд. Очевидно, что если вам приходится часто выполнять одни и те же действия, то использование командных файлов может сэкономить много времени.

Командный файл можно запускать на выполнение точно так же, как и исполняемый файл с расширением `com` или `exe`, в частности, с помощью команды `START`. Для изменения способа выполнения командного файла в Windows необходимо щелкнуть на названии файла в Проводнике (Explorer) правой кнопкой мыши и выбрать пункт **Свойства** (Properties) в появившемся контекстном меню. Затем в открывшемся диалоговом окне (рис. 2.1) можно задать разные параметры для выбранного файла, например, завершение работы Windows перед выполнением данного файла, или будет ли командное окно автоматически закрываться после завершения работы пакетного файла.

После закрытия этого диалогового окна для файла, чьи свойства мы редактировали (`C:\lem.bat`), создастся его информационный файл (Program Information File, PIF); в нашем примере это будет `C:\lem.pif`.

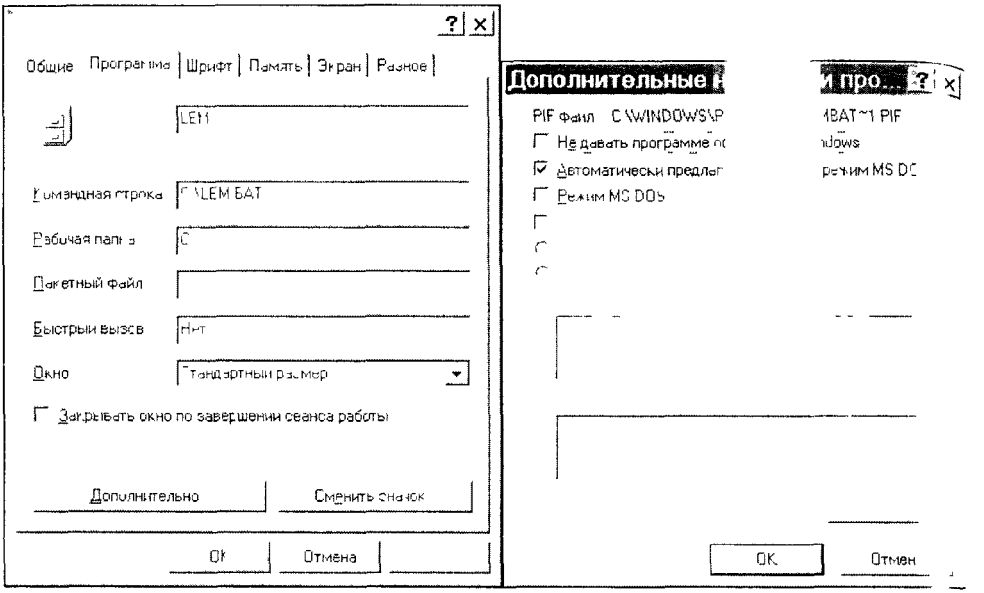


Рис. 2.1. Изменение параметров выполнения командного файла

Для отладки командного файла его можно выполнить по шагам — в этом случае для каждой строки файла будет выводиться запрос на ее выполнение. Переключиться в такой режим построчного выполнения команд можно с помощью команды COMMAND с ключами /Y и /C, после которых указывается имя отлаживаемого файла. Например, для того чтобы выполнить по шагам файл deltmp.bat, нужно воспользоваться следующей командой:

```
COMMAND /Y /C deltmp
```

Напомним, что ключ /C, задающий возврат к исходному командному интерпретатору после выполнения файла deltmp.bat, в COMMAND должен стоять последним, т. е. команда типа

```
COMMAND /C /Y deltmp
```

будет считаться некорректной.

В командных файлах можно использовать комментарии, т. е. строки, которые никак не влияют на выполнение этого файла. Комментарии вносятся с помощью ключевого слова REM, например

```
REM Снимем атрибут "Только чтение"
ATTRIB -R *.tmp
REM Удалим файлы с расширением tmp
DEL *.tmp
```

Вывод сообщений и дублирование команд

По умолчанию команды пакетного файла перед исполнением выводятся на экран, что выглядит не очень эстетично. С помощью команды ECHO OFF можно отключить дублирование команд, идущих после нее (сама команда ECHO OFF при этом все же дублируется). Например,

```
REM Следующие две команды будут дублироваться на экране ...
DIR C:\
ECHO OFF
REM А остальные уже не будут
DIR D:\
```

Для восстановления режима дублирования используется команда ECHO ON. Кроме этого, можно отключить дублирование любой отдельной строки в командном файле, написав в начале этой строки символ @, например:

```
ECHO ON
REM Команда DIR C:\ дублируется на экране
DIR C:\
REM А команда DIR D:\ - нет
@DIR D:\
```

Таким образом, если поставить в самое начало командного файла команду

```
@ECHO OFF
```

то это решит все проблемы с дублированием команд.

Вывести строку сообщения на экран можно с помощью команды

ECHO *сообщение*

Например,

```
@ECHO OFF
ECHO Привет!
```

Команда ECHO. (точка должна следовать непосредственно за словом "ECHO") выводит на экран пустую строку. Например,

```
@ECHO OFF
ECHO Привет!
ECHO.
ECHO Пока!
```

Часто бывает удобно для просмотра сообщений, выводимых из пакетного файла, предварительно полностью очистить экран командой CLS.

Используя описанный в первой главе механизм перенаправления ввода/вывода (символы > и >>), можно направить сообщения, выводимые командой ECHO, в заданный текстовый файл Например,

```
@ECHO OFF
ECHO Привет! > hi txt
ECHO Пока! >> hi.txt
```

Замечание

Напомним, что в случае использования символа > старое содержимое файла для вывода теряется, а при применении символа >> строка сообщения дописывается в конец этого файла

С помощью такого метода можно, скажем, заполнять файлы-протоколы с отчетом о произведенных действиях (см листинг 2 1)

Листинг 2.1. Командный файл, заполняющий файл-протокол report.txt

```
@ECHO OFF
REM Попытка копирования
XCOPY C:\PROGRAMS D:\PROGRAMS /s
REM Добавление сообщения в файл report txt в случае
REM удачного завершения копирования
IF NOT ERRORLEVEL 1 ECHO Успешное копирование >> report.txt
```

Замечание

Использованная в этом примере команда IF будет описана далее в *разд "Переходы и операторы условия в командных файлах"* настоящей главы

Использование параметров командной строки и переменных среды

При запуске пакетных файлов в командной строке можно указывать произвольное число параметров, значения которых допускается использовать внутри файла Это позволяет, например, применять один и тот же командный файл для выполнения команд с различными параметрами

Для доступа к параметрам командноиз строки применяются символы %0, %1, , %9 Вместо %0 подставляется имя выполняемого пакетного файла, а вместо %1, %2, , %9 — значения первых девяти параметров командной строки соответственно Если в командной строке при вызове пакетного файла задано меньше девяти параметров, то "лишние" переменные из %1 – %9 замеща-

ются пустыми строками Рассмотрим следующий пример Пусть имеется командный файл copier.bat такого содержания

```
@ECHO OFF
CLS
ECHO Файл %0 копирует каталог %1 в %2
XCOPY %1 %2 /S
```

Если запустить его из командной строки с двумя параметрами, например

```
copier.bat C:\Programs D:\Backup
```

то на экран выведется сообщение

```
Файл copier.bat копирует каталог C:\Programs в D:\Backup
```

и произойдет копирование каталога C:\Programs со всеми его подкаталогами в D.\Backup

При необходимости можно использовать более девяти параметров командной строки Это достигается с помощью команды SHIFT, которая изменяет значения замещаемых параметров с %0 по %9, копируя каждый параметр в предыдущий, т е значение %1 копируется в %0. значение %2 — в %1 и т д Замещаемому параметру %9 присваивается значение параметра, следующего в командной строке за старым значением %9 Если же такой параметр не задан, то новое значение %9 — пустая строка

Рассмотрим пример Пусть командный файл my.bat вызван из командной строки следующим образом

```
my.bat p1 p2 p3
```

Тогда %0=my bat, %1=p1, %2=p2, %3=p3, параметры %4 — %9 являются пустыми строками После выполнения команды SHIFT значения замещаемых параметров изменятся следующим образом %0=p1, %1=p2, %2=p3, параметры %3 — %9 — пустые строки

Команда, обратная SHIFT (обратный сдвиг), отсутствует После выполнения SHIFT уже нельзя восстановить параметр (%0), который был первым перед сдвигом Если в командной строке задано больше десяти параметров, то команду SHIFT можно использовать несколько раз

С помощью команды SET внутри командных файлов можно работать с переменными среды, в том числе объявлять собственные переменные *(см разд "Работа с переменными среды" гл 1)* Напомним, что в Windows 9x все переменные среды рассматриваются как строки и для получения их значений нужно имя соответствующей переменной заключить в символы % Например,

```
@ECHO OFF
CLS
REM Создание переменной MyVar
SET MyVar=Привет
```

```
REM Изменение переменной
SET MyVar=%MyVar%!
ECHO Значение переменной MyVar: %MyVar%
REM Удаление переменной MyVar
SET MyVar=
```

При запуске такого командного файла на экран выведется строка

```
Значение переменной MyVar: Привет!
```

Внутри командного файла можно использовать и переменные, которые сис тема устанавливает автоматически. Например,

```
@ECHO OFF
CLS
ECHO Каталог Windows: %WinDir%
ECHO Каталог для временных файлов: %TEMP%
```

К сожалению, в командных файлах Windows 9*x* (в отличие от таких файлов Windows NT) нельзя производить арифметических вычислений с перемен ными. Еще раз напомним, что все переменные рассматриваются лишь ка строки, причем над ними можно производить только операцию конкатена ции (склеивания). Для этого нужно в команде `SET` просто написать рядо значения соединяемых переменных. Например:

```
SET A=Раз
SET B=Два
SET C=%A%%B%
```

После выполнения в файле этих команд значением переменной `C` будет яв ляться строка `РазДва`. Не следует для конкатенации использовать знак + т. к. он будет воспринят просто в качестве символа. Например, после запус ка файл следующего содержания

```
SET A=Раз
SET B=Два
SET C=A+B
ECHO Переменная C=%C%
SET D=%A%+%B%
ECHO Переменная D=%D%
```

на экран выведутся две строки:

```
Переменная C=A+B
Переменная D=Раз+Два
```

Все изменения, произведенные в командном файле над переменными сре ды, сохраняются и после завершения работы этого файла (в текущем ко мандном окне, разумеется).

Приостановка выполнения командных файлов

Для того чтобы вручную прервать выполнение запущенного bat-файла, нужно нажать клавиши <Ctrl>+<C> или <Ctrl>+<Break>. Однако часто бывает необходимо программно приостановить выполнение командного файла в определенной строке с выдачей запроса на нажатие любой клавиши. Это делается с помощью команды PAUSE. Перед запуском этой команды полезно с помощью команды ECHO информировать пользователя о действиях, которые он должен произвести. Например:

```
ECHO Вставьте дискету в дисковод A: и нажмите любую клавишу
PAUSE
```

Команду PAUSE обязательно нужно использовать при выполнении потенциально опасных действий (удаление файлов, форматирование дисков и т. п.). Например:

```
ECHO Сейчас будут удалены все файлы в C:\Мои документы!
ECHO Для отмены нажмите Ctrl-C
PAUSE
DEL "C:\Мои документы\*.*"
```

Циклы в командных файлах

В командных файлах для организации циклов используется механизм For ... Next, который есть во многих языках программирования. Основная особенность циклов в командных файлах состоит в том, что при их выполнении не происходит регулярного приращения переменной (счетчика) цикла, а вместо этого для организации счетчика повторяется заданный список значений.

Сам цикл объявляется с помощью команды FOR ... IN ... DO..., которая уже частично рассматривалась в *разд. "Команды сравнения и поиска для файлов" гл. 1*. Синтаксис команды FOR для командных файлов следующий:

```
FOR %%переменная IN (множество) DO команда [параметры]
```

Внимание!

Перед названием переменной должны стоять именно два знака процента (%%), а не один, как это было при использовании команды FOR непосредственно из командной строки.

Сразу приведем пример. Если в командном файле заданы строки

```
@ECHO OFF
FOR %%i IN (Раз,Два,Три) DO ECHO %%i
```

то в результате его выполнения на экране будет напечатано следующее:

```
Раз
Два
Три
```

Параметр *множество* в команде FOR задает одну или более текстовых строк, разделенных запятыми, которые вы хотите обработать с помощью заданной команды. Скобки здесь обязательны. Параметр *команда* [*параметры*] задает команду, выполняемую для каждого элемента *множества*, при этом вложенность команд FOR на одной строке не допускается. Если в строке, входящей во множество, используется запятая, то значение этой строки нужно заключить в кавычки. Например, в результате выполнения файла с командами

```
@ECHO OFF
FOR %%i IN ("Раз,Два",Три) DO ECHO %%i
```

на экран будет выведено

```
Раз,Два
Три
```

Параметр *%%переменная* представляет подставляемую переменную (счетчик цикла), причем здесь могут использоваться только имена переменных, состоящие из одной буквы. При выполнении команда FOR заменяет подставляемую переменную текстом каждой строки в заданном *множестве*, пока команда, стоящая после ключевого слова DO, не обработает все такие строки.

Замечание

Чтобы избежать путаницы с параметрами командного файла %0 — %9, для переменных следует использовать любые символы кроме 0 — 9.

Если ввести в командный файл строку типа

```
FOR %%i IN (1,2,3,4,5) DO ECHO %%i
```

то мы получим подобие традиционного оператора For...Next. Однако вновь подчеркнем, что элементами множества (1,2,3,4,5) можно пользоваться только как строками — в пакетных файлах Windows 9*x* нельзя производить арифметические вычисления.

Параметр *множество* в команде FOR может также представлять одну или несколько групп файлов. Например, чтобы вывести в файл список всех файлов с расширениями txt и prn, находящихся в каталоге C:\TEXT, без использования команды DIR, можно использовать командный файл следующего содержания:

```
@ECHO OFF
FOR %%f IN (C:\TEXT\*.txt C:\TEXT\*.prn) DO ECHO %%f >> list.txt
```

При таком использовании команды FOR процесс обработки продолжается, пока не обработаются все файлы (или группы файлов), указанные во *множестве*.

Вызов внешних командных файлов

Из одного командного файла можно вызвать другой, просто указав его имя. Например:

```
@ECHO OFF
CLS
REM Вывод списка log-файлов
DIR C:\*.log
REM Передача выполнения файлу f.bat
f.bat
COPY A:\*.* C:\
PAUSE
```

Однако в этом случае после выполнения вызванного файла управление в вызывающий файл не передается, т. е. в приведенном примере команда

```
COPY A:\*.* C:\
```

(и все следующие за ней команды) никогда не будет выполнена.

Для того чтобы вызвать внешний командный файл с последующим возвратом в первоначальный файл, нужно использовать специальную команду

```
CALL файл
```

Например:

```
@ECHO OFF
CLS
REM Вывод списка log-файлов
DIR C:\*.log
REM Передача выполнения файлу f.bat
CALL f.bat
COPY A:\*.* C:\
PAUSE
```

В этом случае после завершения работы файла `f.bat` управление вернется в первоначальный файл на строку, следующую за командой CALL (в нашем примере это команда `COPY A:\*.* C:\`).

Внутри пакетного файла нельзя явным образом использовать подпрограммы, однако можно создать несколько пакетных файлов: один основной и один или несколько вспомогательных, каждый из которых может обрабатывать параметры командной строки. Так как после вызова с помощью коман-

ды CALL вспомогательного файла из основного управление возвращается на следующую инструкцию основного файла, то можно считать, что таким образом мы вызывали подпрограмму.

Этим приемом можно пользоваться, скажем, при работе с оператором FOR...IN...DO, который, как мы видели, задает цикл только для одной команды. Рассмотрим пример.

Пусть в командном файле proc.bat записаны следующие строки:

```
@ECHO OFF
ECHO Записываем файл %1.txt
ECHO Параметр вызова: %1 > %1.txt
```

В другом пакетном файле main.bat введем:

```
@ECHO OFF
FOR %%i IN (Раз,Два,Три) DO CALL proc.bat %%i
```

Если запустить файл main.bat на исполнение, то это приведет к троекратному выполнению файла proc.bat. В результате парной работы этих двух командных файлов создадутся три текстовых файла: Раз.txt, Два.txt и Три.txt. При этом в Раз.txt будет записана строка Параметр вызова: Раз, в Два.txt — Параметр вызова: Два, в Три.txt — Параметр вызова: Три.

Переходы и операторы условия в командных файлах

Командный файл может содержать метки и команды GOTO перехода к этим меткам. Любая строка, начинающаяся с двоеточия :, воспринимается при обработке командного файла как метка. Имя метки задается набором символов, следующих за двоеточием до первого пробела или конца строки. Приведем пример.

Пусть имеется командный файл следующего содержания:

```
@ECHO OFF
COPY %1 %2
GOTO Label1
ECHO Эта строка никогда не выполнится
:Label1
REM Продолжение выполнения
DIR %2
```

После того, как в этом файле мы доходим до команды

```
GOTO Label1
```

его выполнение продолжается со строки

```
REM Продолжение выполнения
```

Команда GOTO часто используется вместе с командой IF для выполнения перехода по условию.

С помощью команды IF в пакетных файлах можно выполнять обработку условий трех типов. При этом, если заданное в команде условие истинно, система выполняет следующую за условием команду, в противном случае эта команда игнорируется.

Первый тип условия используется обычно для проверки значения переменной. Для этого применяется следующий синтаксис команды IF:

```
IF [NOT] строка1==строка2 команда
```

Условие *строка1==строка2* (здесь необходимо писать именно два знака равенства) считается истинным при точном совпадении обеих строк. Параметр NOT указывает на то, что заданная команда выполняется лишь в том случае, когда сравниваемые строки не совпадают.

Строки могут быть литеральными или представлять собой значения переменных (например, %1 или %TEMP%). Кавычки для литеральных строк не требуются. Например,

```
IF %1==%2 ECHO Параметры совпадают!
IF %1==Петя ECHO Привет, Петя!
```

Отметим, что при сравнении строк, заданных переменными, следует проявлять определенную осторожность. Дело в том, что значение переменной может оказаться пустой строкой, и тогда может возникнуть ситуация, при которой выполнение командного файла аварийно завершится. Например, если вы не определили с помощью команды SET переменную MyVar, а в файле имеется условный оператор типа

```
IF %MyVar%==C:\ ECHO Ура!!!
```

то в процессе выполнения вместо %MyVar% подставится пустая строка и возникнет синтаксическая ошибка. Такая же ситуация может возникнуть, если одна из сравниваемых строк является значением параметра командной строки, т. к. этот параметр может быть не указан при запуске командного файла. Поэтому при сравнении строк нужно приписывать к ним в начале какой-нибудь символ, например:

```
IF -%MyVar%==-C:\ ECHO Ура!!!
```

С помощью команд IF и SHIFT можно в цикле обрабатывать все параметры командной строки файла, даже не зная заранее их количества. Например, командный файл primer.bat, приведенный в листинге 2.2, выводит на экран имя запускаемого файла и все параметры командной строки.

Листинг 2.2. Вывод на экран всех параметров командной строки (primer.bat

```
@ECHO OFF
ECHO Выполняется файл. %0
ECHO
ECHO Файл запущен со следующими параметрами...
REM Начало цикла
:BegLoop
IF "%1"=="" GOTO ExitLoop
ECHO %1
REM Сдвиг параметров
SHIFT
REM Переход на начало цикла
GOTO BegLoop
:ExitLoop
REM Выход из цикла
ECHO
ECHO Все.
```

Если запустить primer.bat с четырьмя параметрами:

```
primer.bat А Б В Г
```

то в результате выполнения на экран выведется следующая информация

```
Выполняется файл: primer.bat

Файл запущен со следующими параметрами:
А
Б
В
Г

Все.
```

Второй способ использования команды IF — это проверка существования заданного файла. Синтаксис для этого случая имеет вид

```
IF [NOT] EXIST файл команда
```

Условие считается истинным, если указанный файл существует. Кавычки для имени файла не требуются. Соответствующий пример командного файла, в котором используется такой вариант команды IF, приведен в листинге 2.3.

Листинг 2.3. Проверка наличия файла, указанного в качестве параметра командной строки

```
@ECHO OFF
IF "%1"=="" Goto NoFileSpecified
if not exist %1 goto FileNotExist
```

```
rem Вывод сообщения о найденном файле
echo Файл '%1' успешно найден.
goto Exit

:NoFileSpecified
rem Файл запущен без параметров
echo В командной строке не указано имя файла.
Goto Exit

:FileNotExist
rem Параметр командной строки задан, но файл не найден
echo Файл '%1' не найден.

:Exit
rem Выход из файла
```

Ключ NOT позволяет проверить отсутствие заданного файла Например,

```
IF NOT EXIST C:\autoexec.bat ECHO У вас нет файла autoexec.bat
```

Наконец, третий способ использования команды IF — это проверка кода завершения (кода выхода) предыдущей команды Синтаксис для IF в этом случае имеет следующий вид

```
IF [NOT] ERRORLEVEL число команда
```

Здесь условие считается истинным, если последняя запущенная команда или программа завершилась с кодом возврата, равным либо превышающим указанное число

Составим, например, командный файл mycopy bat, который бы копировал файл my txt на диск C: без вывода на экран сообщений о копировании, а в случае возникновения ошибки выдавал предупреждение (см листинг 2 4)

Листинг 2.4. Файл mycopy.bat, осуществляющий контроль ошибок копирования

```
@echo off
xcopy my.txt C.\ > NUL
REM Проверка кода завершения копирования
if errorlevel 1 goto ErrOccurred
echo Копирование выполнено без ошибок.
goto EndBatch

:ErrOccurred
echo При выполнении команды XCOPY возникла ошибка!

:EndBatch
```

Диалоговые командные файлы

В командном файле во время его выполнения нельзя с помощью стандартных команд обеспечить ввод строки с клавиатуры, т. е. нельзя обеспечить полноценный диалог с пользователем. Единственное, что Windows 9*x* может предложить в этом плане — это команда CHOICE, которая выводит пользователю заданную подсказку и ждет, пока он выберет нужный вариант из указанного набора клавиш. Эту команду можно использовать только в командных файлах. Ее синтаксис имеет вид:

```
CHOICE [/C[:]варианты] [/N][/S][/T[:]c,nn] [текст]
```

Ключ /C:*варианты* задает варианты ответа пользователя, т. е. допустимые в подсказке клавиши. По умолчанию строка включает два варианта YN. При выводе на экран варианты будут разделяться запятыми, заключаться в квадратные скобки ([]) и сопровождаться вопросительным знаком. Например, если в командном файле вы укажете команду:

```
CHOICE /c:ync
```

то пользователь на экране увидит следующее: `[Y,N,C]?`

Параметр *текст* (кавычки задавать не нужно) задает текст, который вы хотите выводить перед подсказкой. Скажем, в результате выполнения такой команды:

```
choice /c:ync Yes, No, or Cancel
```

пользователю выводится:

```
Yes, No, or Cancel [Y,N,C]?
```

Если вы не задаете *текст*, то команда CHOICE выводит на экран только подсказку.

Использование ключа /N приводит к тому, что ни сами варианты, ни знак вопроса в строке приглашения не отображаются. Однако текст перед подсказкой выводится. При задании /N указанные клавиши все равно будут доступны.

Ключ /S позволяет учитывать регистр символов. Если /S не задан, то для любых используемых пользователем клавиш будет восприниматься как верхний, так и нижний регистры (т. е. y и Y будут восприниматься одинаково).

Если задан ключ /T[:]c,nn, то команда CHOICE перед использованием заданной клавиши по умолчанию делает паузу в течение заданного числа секунд. При использовании /T указываются следующие значения:

- c — определяет символ, который спустя nn секунд будет задаваться по умолчанию (этот символ должен быть в наборе символов, заданном в ключе /C);

- nn — задает продолжительность паузы в секундах (допустимые значения nn лежат в диапазоне от 0 до 99, при этом если задается 0, то перед назначением по умолчанию будет неограниченная пауза).

После выполнения команды CHOICE переменная ERRORLEVEL приобретает значение, равное номеру выбранного варианта ответа. Это позволяет использовать команду IF для того, чтобы узнать, какой именно из предложенных вариантов был выбран пользователем. Если CHOICE обнаруживает состояние ошибки, то возвращается значение 255. Если пользователь нажал клавиши <Ctrl>+<Break> или <Ctrl>+<C>, CHOICE возвращает значение, равное 0.

Например, если вы задали команду:

```
choice /c:ync /t:N,10
```

то пользователь видит на экране:

```
[Y,N,C]?
```

Если в течение 10 секунд пользователь не нажмет клавишу, то CHOICE по истечении указанного времени выбирает N и возвращает значение ERRORLEVEL, равное 2. При нажатии соответствующей клавиши до истечения 10 секунд CHOICE возвращает значение, соответствующее выбору пользователя.

В качестве примера, показывающего, как с помощью команды CHOICE можно в командном файле организовать пользовательское меню, рассмотрим файл mymenu.bat (см. листинг 2.5).

Листинг 2.5. Командный файл mymenu.bat, создающий меню на экране

```
@ECHO OFF
Rem Вывод меню на экран
echo Пункты меню:
echo.
echo А  — Пункт А
echo Б  — Пункт Б
echo В  — Пункт В
echo.

REM Вывод подсказки для ввода
choice /C:АБВ Выберите пункт меню
if errorlevel 3 goto ChoiceV
if errorlevel 2 goto ChoiceB
If errorlevel 1 goto ChoiceA
echo Выбор не был сделан.
goto Done

:ChoiceA
echo Выбран пункт А!
Goto Done
```

```
:ChoiceB
echo Выбран пункт Б!
Goto Done

:ChoiceV
echo Выбран пункт В!
Goto Done
:Done
```

Интерактивная загрузка операционной системы

В *разд. "Команды конфигурации системы" гл. 1* мы уже описывали процесс загрузки Windows 9x. На самом деле, существуют две возможности в той или иной степени управлять процессом загрузки: загрузочное меню Windows и загрузочное меню MS-DOS.

Загрузочное меню Windows появляется на экране, если Windows не смогла успешно загрузиться или не была полностью выгружена при последнем выключении компьютера. Главное назначение этого меню — обеспечить доступ к командной строке и безопасному режиму Windows. Кроме того, это загрузочное меню может быть вызвано пользователем — для этого надо нажать клавишу <F8> во время загрузки (до появления заставки Windows). Меню может включать в себя восемь (или меньше) пунктов. Количество пунктов меню зависит от настроек в файле `msdos.sys` *(см. разд. "Команды конфигурации системы" гл. 1)*. Итак, по умолчанию пользователю предлагаются следующие пункты.

1. **Normal** (Обычный). Режим обычной загрузки Windows.
2. **Logged** (С записью протокола загрузки). Все шаги процесса загрузки будут записываться в файл `bootlog.txt`, находящийся в корневом каталоге загрузочного диска. После загрузки можно просмотреть этот файл и выяснить, все ли драйверы и задачи были успешно загружены.
3. **Safe Mode** (Безопасный режим). Используется при загрузке Windows в режиме VGA (640×480) и без сетевой поддержки.
4. **Safe Mode with Network Support** (Безопасный режим с сетевой поддержкой). Загрузка происходит в режиме VGA (640×480), но с подключением сетевых драйверов (если они установлены).
5. **Step-by-step Confirmation** (Пошаговое подтверждение). То же самое, что пункт 1, за исключением того, что Windows запрашивает подтверждение перед загрузкой каждого драйвера. Пошаговые подтверждения включаются и отключаются нажатием клавиш <Shift>+<F8>, поэтому этот режим можно использовать совместно с любым другим.

6. **Command Prompt Only** (Командная строка). В этом случае происходит загрузка в режиме MS-DOS, т. е. после выполнения файлов `config.sys` и `autoexec.bat` не происходит дальнейшего запуска графической системы Windows. Нажав здесь или при включении компьютера клавиши <Shift>+<F5>, можно загрузиться в режиме MS-DOS без выполнения файлов `config.sys` и `autoexec.bat`.
7. **Safe Mode Command Prompt Only** (Безопасный режим командной строки). То же самое, что пункт 6, за исключением того, что файлы `config.sys` и `autoexec.bat` пропускаются.
8. **Previous version of MS-DOS** (Предыдущая версия MS-DOS). Если вы установили Windows поверх старой версии MS-DOS и указали, что ее следует оставить на вашей системе, то, выбрав этот пункт меню, вы загрузите старую версию MS-DOS.

Второй тип загрузочного меню впервые появился в MS-DOS 5.0, с его помощью можно задать любое число различных конфигураций системы. Например, можно задать варианты загрузки в режиме MS-DOS, загрузку Windows, загрузку Windows в безопасном режиме или загрузку в MS-DOS со специальными драйверами. Особенно это бывает удобно, если вы часто используете приложения MS-DOS, для которых требуются специальные настройки.

Для создания загрузочного меню MS-DOS необходимо специальным образом разбить файл `config.sys` на разделы (блоки), указать там специальные команды, а затем проанализировать в файле `autoexec.bat` сделанный пользователем выбор. Опишем этот процесс более подробно.

Для создания меню файл `config.sys` должен, во-первых, содержать блок меню, который начинается с ключевого заголовка `[Menu]`. Блок меню — это набор команд определения меню, которые начинаются с заголовка раздела (имени раздела в квадратных скобках). Команда `MENUITEM` определяет пункт меню:

```
MENUITEM=имя_блока[,текст_меню]
```

Параметр *имя_блока* задает имя соответствующего блока конфигурации, который должен определяться где-либо в файле `config.sys`. Если при запуске пользователь выбирает определенный пункт меню, то система выполняет команды в соответствующем блоке конфигурации, а также команды в начале `config.sys` и все команды в блоках конфигурации с заголовком `[Common]`.

Имя блока может включать в себя до 70 символов и содержать большинство печатаемых символов, кроме пробелов, обратной и прямой косой черты, запятой, точки с запятой, знака равенства или квадратных скобок. Параметр *текст_меню* задает текст, который выводится для данного пункта меню. Если текст не задается, то в качестве элемента меню выводится имя блока. Текст меню может быть длиной до 70 символов и содержать любые символы.

Например, чтобы создать загрузочное меню с двумя вариантами выбо файл config.sys должен быть примерно таким:

```
[Menu]
MenuItem=Variant1, First variant
MenuItem=Variant2, Second variant
[Variant1]
Команды для первого варианта
[Variant2]
Команды для второго варианта
[Common]
Команды, которые нужно выполнять вне зависимости
от выбранного пункта меню
```

Тогда при запуске Windows на экране появится меню вида

```
Microsoft Windows 95 Startup Menu
=================================
1. First variant
2. Second variant
Enter a choice: 1
```

С помощью команды MENUDEFAULT можно задать используемый по умолчанию элемент меню и установить, если нужно, значение тайм-аута. Если эта команда не задана, то по умолчанию выбирается первый элемент.

Синтаксис команды имеет вид:

```
MENUDEFAULT=имя_блока[,тайм_аут]
```

Например:

```
[Menu]
MENUITEM=Variant1, First variant
MENUITEM=Variant2, Second variant
MENUDEFAULT=Variant2,15
[Variant1]
Команды для первого варианта
[Variant2]
Команды для второго варианта
```

При загрузке используемый по умолчанию пункт меню подсвечивается, номер его выводится после подсказки Enter a choice Параметр *тайм_а* определяет, сколько секунд должна ждать система перед запуском компью тера с конфигурацией по умолчанию. Меню для нашего примера буде иметь следующий вид:

```
Microsoft Windows 95 Startup Menu
==================================
1. First variant
2. Second variant
Enter a choice: 2   Time remaining: 15
```

Если значение параметра *тайм_аут* не задано, то Windows просто ожидает нажатия клавиши <Enter>. Вы можете задать значение тайм-аута от 0 до 90 секунд, причем 0 определяет автоматически выбор конфигурации по умолчанию.

В загрузочном меню можно определить одно или несколько подменю. Для этого используется команда SUBMENU. Синтаксис этой команды тот же, что и в команде MENUITEM:

```
SUBMENU =имя_блок[,текст_меню]
```

Например,

```
[Menu]
MENUITEM=Variant1, First variant
MENUITEM=Variant2, Second variant
SUBMENU=Variant3, Advanced variant
[Variant1]
Команды для первого варианта
[Variant2]
Команды для второго варианта
[Variant3]
MENUITEM=Variant4, First advanced variant
MENUITEM=Variant5, Second advanced variant
```

Здесь при выборе пункта Advanced variant будет выведено такое меню:

```
Microsoft Windows 95 Startup Menu
==================================
1. First advanced variant
2. Second advanced variant
Enter a choice: 1
```

Вернуться в главное меню из подменю уже нельзя.

С помощью команды INCLUDE можно включать содержимое одного блока меню в другой. Синтаксис этой команды имеет вид:

```
INCLUDE=имя_блока
```

Здесь параметр *имя_блока* определяет блок, который нужно включить. Например:

```
[Menu]
MENUITEM=Variant1, First variant
MENUITEM=Variant2, Second variant
[Variant1]
Команды для первого варианта
[Variant2]
INCLUDE=Variant1
Команды для второго варианта
```

В этом случае команды для первого варианта будут выполнены и при выборе второго варианта.

После того как пользователь сделал свой выбор в загрузочном меню, сформированном в файле `config.sys`, мы можем проанализировать в файле `autoexec.bat` этот выбор и в зависимости от него выполнить те или иные действия. Дело в том, что после того как выполнятся все команды из файла `config.sys`, создается переменная среды `CONFIG`, значением которой является имя выбранного пункта загрузочного меню. Таким образом, для нашего примера файл `autoexec.bat` может иметь структуру следующего типа:

```
@ECHO OFF
IF "%CONFIG%"=="Variant1" GOTO Var1
IF "%CONFIG%"=="Variant2" GOTO Var2
:Var1
Команды для первого варианта
:Var2
Команды для второго варианта
```

Пример использования командных файлов

Windows 98 имеет в своем составе адресную строку, которая фактически выполняет те же функции, что и пункт **Выполнить** (Run) в меню **Пуск** (Start): используя эту панель можно не только открывать Web-страницы, но и запускать любую программу. Однако из адресной строки нельзя напрямую запускать внутренние команды Windows (например, `DIR` или `COPY`), ведь для их исполнения необходимо предварительно загрузить в оперативную память командный интерпретатор. Можно, конечно, для этой цели использовать команду `COMMAND` с ключами `/C` или `/K`, но это очень неудобно.

Рассмотрим в качестве примера использования командных файлов, каким образом можно приписать адресной строке почти все функции командной строки (эта идея позаимствована из книги [7]). Таким образом, вы получите на своем рабочем столе инструмент, с помощью которого можно открывать папки и Web-страницы, запускать программы и команды, просто вводя их

название и нажимая клавишу <Enter>. Кроме того, у вас будет вестись история выполняемых команд.

Итак, сначала выведите адресную строку на экран (если у вас ее еще нет). Для этого щелкните правой кнопкой мыши на любом месте пустого пространства панели задач, выберите пункт **Панели инструментов** (Tools bar), затем **Адресная строка** (Address bar). На панели задач появится адресная строка (рис. 2.2). С помощью режима перетаскивания можно изменять размер адресной строки и перемещать ее по панели задач.

Рис. 2.2. Адресная строка на панели задач

Затем создайте командный файл +.bat, приведенный в листинге 2.6, и сохраните этот файл в каталоге, входящем в системный путь, например, C:\WINDOWS\COMMAND.

Листинг 2.6. Файл +.bat

```
@echo off
if "%1"=="" exit
if exist %TEMP%\temp.bat del %TEMP%\temp.bat
ECHO %1 %2 %3 %4 %5 %6 %7 %8 %9 > %TEMP%\temp.bat
CALL %TEMP%\temp.bat
if exist %TEMP%\temp.bat DEL %TEMP%\temp.bat
```

Теперь для запуска внутренней команды Windows из адресной строки достаточно поставить перед этой командой знак + и пробел, например:

```
+ DIR %WinDir%
```

Разберемся теперь в деталях. Созданный командный файл +.bat считывает набранную вами команду со всеми ее параметрами и ключами (их количество не должно превышать восьми) и записывает эту команду с помощью команды ECHO в новый временный файл temp.bat. Затем файл temp.bat исполняется, что и обеспечивает выполнение введенной в адресной строке команды со всеми параметрами и ключами (их количество не должно превышать восьми). По окончании своей работы temp.bat удаляется.

Конечно, при таком подходе в течение сеанса выполняется всего одна команда Windows, что означает, в частности, что понятие "текущий каталог" не имеет смысла и все команды нужно вводить с полными путями к файлам.

Заключение

С помощью командных файлов можно автоматизировать выполнение несложных, но часто повторяющихся рутинных задач, например, резервного копирования важных данных или подключения сетевых ресурсов.

Командные файлы в Windows 9*x* обладают следующими возможностями:

- использование в командах внутри файла замещаемых параметров командной строки и переменных среды;
- вывод строк текста на экран и во внешний файл;
- вызов внешних командных файлов (симуляция подпрограмм);
- использование циклов для выполнения одной команды, в том числе для обработки множества файлов;
- использование (ограниченное) оператора условия `IF`;
- переход на заданную команду внутри текущего файла;
- возможность выбора пользователем во время выполнения файла одного из предложенных вариантов.

Очевидны также и следующие недостатки командных файлов, которые не позволяют применять их для выполнения задач, где требуется использование более-менее сложного алгоритма:

- недостаточная гибкость условного оператора `IF` (сравнение производится только на равенство строк);
- невозможность проводить арифметические вычисления;
- отсутствие функций для работы со строками;
- невозможность считать построчно текстовый файл и выполнить какие-либо действия с его строками.

Эти недостатки исправлены (в той или иной степени) в Windows NT.

Упражнения

Для проверки знаний, полученных при прочтении настоящей главы, читателям предлагается выполнить следующие упражнения.

1. Пусть имеется текстовый файл `protokol.txt`, в котором хранится журнал обработанных файлов в следующем формате:

```
Имя: klop04.txt    Дата: 02.01.2001  Время: 14:50
Имя: klop23.txt    Дата: 03.02.2001  Время: 23:50
Имя: astra12.txt   Дата: 02.01.2001  Время: 12:00
Имя: soft.txt      Дата: 10.01.2000  Время: 13:00
```

Слово Дата здесь начинается в каждой строке с двадцатой позиции. Необходимо написать командный файл, с помощью которого сделать выборку из этого файла (т. е. создать новый текстовый файл с нужной информацией) за заданный месяц (ММ) и год (ГГГГ) в файл ММГГГГ.txt, сформированный файл упорядочить по дате обработки. Нужные месяц и год указать как параметры командной строки.

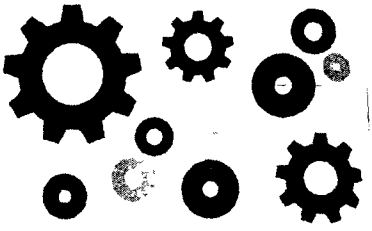
2. Написать командный файл, который будет копировать из текущего каталога все файлы с расширением txt, кроме одного файла, указанного в качестве второго параметра командной строки, в каталог, указанный первым параметром. Если имя каталога, в который должно производиться копирование, не задано, то вывести сообщение об этом и прервать выполнение файла.

3. Задание аналогично второму упражнению. Дополнительные требования: а) переписывать только те файлы, которые новее одноименных в каталоге-приемнике, б) не прерывать копирование в случае возникновения ошибки, в) записывать в файл logcopy.txt имя каждого копируемого файла и результат выполнения операции для него. Другими словами, файл logcopy.txt должен быть примерно таким:

```
Успешно: a.txt
Ошибка : b.txt
Успешно: c.txt
```

4. Создать командный файл, который выводил бы содержимое каталога, указанного в качестве параметра командной строки, причем пользователю должна быть предоставлена возможность выбора с помощью меню устройства для вывода: на экран (информация выводится по одному экрану), в текстовый файл C:\catalog.txt или на принтер.

ГЛАВА 3



Команды Windows NT

Ядро операционной системы Windows NT, в отличие от Windows 9*x*, уже никак не опирается на MS-DOS, в его основе лежат другие принципы. Однако в Windows NT по-прежнему встроена виртуальная машина MS-DOS (NT Virtual DOS Machine, NTVDM), которая предоставляет режим командной строки для ввода и выполнения утилит и пакетных файлов.

Замечание

Говоря об операционной системе Windows NT, мы подразумеваем Windows NT Workstation 4.0 и Windows NT Server 4.0, т. к. в дальнейшем рассматриваются только команды, которые имеются в обеих этих операционных системах.

Фактически в составе Windows NT имеются два командных интерпретатора. Это обычный интерпретатор команд MS-DOS, представленный файлом `command.com`, и специальный интерпретатор команд Windows NT `cmd.exe`, обладающий гораздо более мощными возможностями. Основные свойства интерпретатора `command.com` были рассмотрены в предыдущих главах и здесь описываться не будут; говоря далее о командном интерпретаторе Windows NT, мы будем подразумевать `cmd.exe`.

В Windows NT уже нельзя перезагрузить компьютер в режиме MS-DOS для того, чтобы работать только с командной строкой. В остальном же доступ к командной строке и работа с ней производится так же, как в Windows 9*x*: для открытия нового командного окна необходимо запустить файл `cmd.exe`. Обычно для этого в меню **Пуск/Программы** (Start/Programs) имеется ярлык **Командная строка** (Command Prompt).

Рассмотрим новые возможности, которые предоставляет Windows NT для работы с командной строкой и командными файлами.

Командный интерпретатор *cmd.exe*

Запустив программу cmd.exe, мы получаем новое окно (рис. 3.1), в котором можно пользоваться командной строкой.

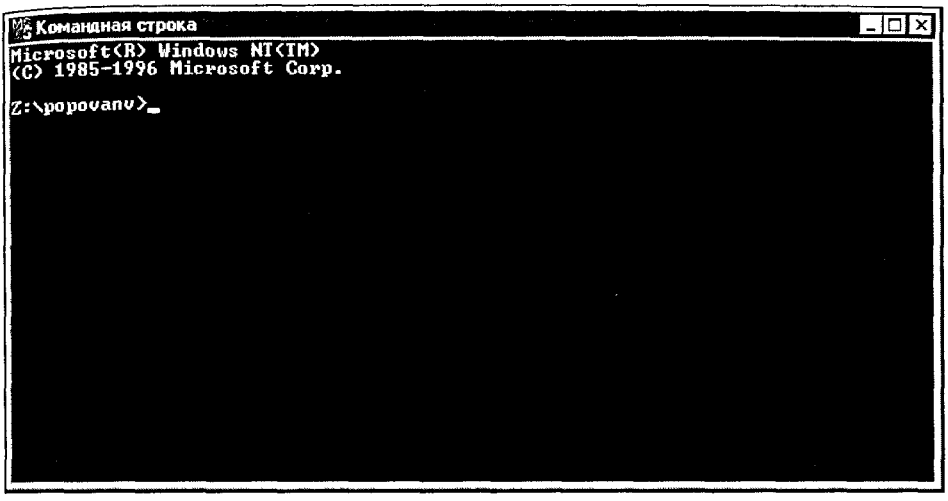


Рис. 3.1. Командное окно в Windows NT Workstation 4.0

Замечание

Отметим, что в Windows NT файл cmd.exe, как и другие исполняемые файлы, соответствующие внешним командам операционной системы, находятся в каталоге %SystemRoot%\SYSTEM32. Значением переменной среды %SystemRoot% является системный каталог Windows NT, обычно C:\WINNT.

Как и в Windows 9*x*, используя утилиту CMD в уже открытом командном окне, мы можем запустить еще несколько версий командного интерпретатора. Для окончания работы текущей версии используется команда EXIT.

Интерпретатор команд cmd.exe имеет два режима работы: стандартный и расширенный, обеспечивающий более богатые возможности программирования команд оболочки. Расширенную версию интерпретатора используют следующие команды: DEL (ERASE), COLOR, CD (CHDIR), MD (MDIR), PROMPT, PUSHD, POPD, SET, SETLOCAL, ENDLOCAL, IF, FOR, CALL, SHIFT, GOTO, START, ASSOC и FTYPE. Рассмотрим подробнее все эти команды.

Команда *CMD*

Синтаксис команды CMD имеет вид:

CMD [/X|/Y][/A|/U][/Q][/T:*цвет*][[/C|/K] *команда*]

Ключи /X и /Y определяют, какая версия командного интерпретатора буде использоваться: стандартная или расширенная. Ключ /X подключает расши ренную версию, а ключ /Y отключает расширения командного интерпрет тора. По умолчанию расширения разрешены, хотя в случае необходимос можно запретить их использование. Для этого нужно установить значение в следующий ключ системного реестра:

```
HKEY_CURRENT_USER\Software\Microsoft\Command Processor\EnableExtensions
```

Ключи /A и /U отвечают за кодировку, с помощью которой происходит вывод информации в командном окне. Ключ /A позволяет выводить информацию символами ANSI, ключ /U — символами Unicode.

Ключ /Q отвечает за режим дублирования команд на экране (Echo). Если указан этот ключ, то команды дублироваться не будут (такого же эффекта можно добиться с помощью команды ECHO OFF).

С помощью ключа /T:*цвет* можно изменять цвет символов и фона на экране. Атрибуты цветов задаются в виде двух шестнадцатеричных чисел — первое задает цвет фона, а второе определяет цвет текста. Каждому числу соответствует следующий цвет:

0 = Черный	8 = Серый
1 = Синий	9 = Светло-синий
2 = Зеленый	A = Светло-зеленый
3 = Голубой	B = Светло-голубой
4 = Красный	C = Светло-красный
5 = Лиловый	D = Светло-лиловый
6 = Желтый	E = Светло-желтый
7 = Белый	F = Ярко-белый

Например, команда:

```
CMD /T:FC
```

задает вывод светло-красного текста на ярко-белом фоне.

Ключи /C и /K имеют те же значения, что и для интерпретатора command.com в Windows 9*x*. При наличии одного из этих ключей (напомним, что они должны стоять последними) происходит запуск новой копии командного интерпретатора и выполнение команды, указанной параметром *команда*. Если задан ключ /C, то работа новой копии интерпретатора прекращается, если ключ /K — продолжается.

Команда *PROMPT*

Как и в Windows 9*x*, за текст, отображаемый в приглашении командной строки, отвечает команда

```
PROMPT [текст]
```

В командном окне Windows NT приглашение может включать обычные символы и специальные коды, приведенные в табл. 3.1.

Таблица 3.1. *Коды команды* `PROMPT`

Код	Символ	Код	Символ
`$C`	((левая круглая скобка)	`$A`	& (амперсанд)
`$F`	) (правая круглая скобка)	`$S`	(пробел)
`$Q`	= (знак равенства)	`$$`	$ (символ доллара)
`$T`	Текущее время	`$D`	Текущая дата
`$P`	Текущие диск и путь	`$V`	Номер версии DOS или Windows
`$N`	Текущий диск	`$G`	> (знак "больше")
`$L`	< (знак "меньше")	`$B`	| (вертикальная черта)
`$H`	Backspace (удаление предыдущего символа)	`$E`	Код Escape (ASCII 27)
`$_`	Возврат каретки и перевод строки		

Если включена расширенная обработка команд (`CMD /X`), то команда `PROMPT` поддерживает еще два дополнительных специальных кода. Код `$+` позволяет отображать нужное число знаков плюс (+) в зависимости от текущей глубины стека каталогов, созданного командой `PUSHD`, по одному знаку на каждый сохраненный путь (команда `PUSHD` подробно описывается в соответствующем разделе гл. 4). С помощью кода `$M` можно отобразить полное имя сетевого диска, связанное с именем текущего диска. Например, если диск `E:` подключен к сетевому ресурсу `\\Server1\Programs`, то команда

```
PROMPT $M$P$G
```

задаст следующий формат приглашения командной строки:

```
\\Server1\Programs E:\>
```

Если текущий диск не является сетевым, то символу `$M` будет соответствовать пустая строка.

Команда *TITLE*

В Windows NT, в отличие от Windows 9*x*, в любой момент заголовок командного окна легко можно изменить. Это делается с помощью команды TITLE, в качестве параметра которой используется новый заголовок, например:

```
TITLE Это новый заголовок!
```

Замечание

Отметим, что текст нового заголовка, используемый в качестве параметра в команде TITLE, записывается без кавычек.

После задания заголовка окна он может быть изменен только повторным вызовом команды TITLE.

Команда TITLE может быть полезна для установки заголовка окна пакетных программ, т. к. в отличие от команды ECHO с ее помощью можно информировать пользователя о действиях, происходящих в командном файле, даже если командное окно, в котором выполняется этот файл, минимизировано и строки, выводимые на экран с помощью ECHO, не видны. Например:

```
@ECHO OFF
TITLE Копируются файлы...
COPY \\ Server1\Share\*.xls C:\Users\Common\*.xls
ECHO Копирование закончено.
TITLE Процесс завершен
```

В этом случае во время копирования файлов заголовком окна будет строка `Копируются файлы...`, а после завершения копирования — строка `Процесс завершен`.

Новые возможности команд Windows NT

Командный интерпретатор в Windows NT поддерживает почти все возможности интерпретатора команд, имевшегося в MS-DOS и Windows 9*x*. Кроме этого он добавляет следующие новые возможности:

1. В Windows NT по сравнению с Windows 9*x* включено больше команд, имевшихся в MS-DOS 6.2. Примером таких практически полезных команд являются REPLACE и TREE, которые будут подробно рассматриваться далее в *разд. "Изменения в командах для работы с файловой системой"* этой главы. Напомним, что в приложении 1 приведены сведения о том, какие именно команды применяются в Windows 9*x* и Windows NT.
2. Во многих командах произведены изменения и улучшения, добавлены новые ключи. Такими доработанными командами являются, например, часто используемые утилиты DEL, DIR и XCOPY. Отдельно упомянем

команду START, в которой стало возможным задавать приоритет исполняемой задачи (эта команда подробно рассматривается далее в *разд. "Запуск программ и документов"* этой главы). Также существенно увеличено число команд для работы с локальной сетью.

3. В Windows NT появляется много принципиально новых возможностей, доступных при работе с командной строкой, например:
   - запуск программ и команд операционной системы в заданное время;
   - ввод в командной строке сразу нескольких команд с возможностью управления запуском определенной команды в зависимости от результатов выполнения предыдущих (условное выполнение команд);
   - поиск строк в текстовых файлах с использованием регулярных выражений;
   - задание и корректировка из командной строки ассоциаций, связанных с файлами определенного типа.
4. Также в Windows NT появляются команды для работы с файловой системой NTFS (New Technology File System), позволяющие, например, просматривать и изменять таблицы контроля доступа (ACL, Access Control List) к файлам. Использование расширенной версии командного интерпретатора cmd.exe обеспечивает дополнительные функции во многих командах и делает язык командных файлов более гибким и приспособленным для решения практических задач.

В табл. 3.2 кратко описаны некоторые команды, не имеющие аналогов в MS-DOS и Windows 9*x*.

Таблица 3.2. *Новые команды Windows NT*

Команда	Описание
AT	Запуск программ и команд в заданное время
CACLS	Просмотр или изменение таблиц контроля доступа (ACL) файлов
CONVERT	Преобразование томов с файловой системой FAT в тома с файловой системой NTFS
DISKPERF	Запуск или остановка системного счетчика дисковой производительности
DOSONLY	Разрешение запуска только приложений MS-DOS с командной строки интерпретатора command.com
ECHOCONFIG	Вывод на экран сообщений подсистемы MS-DOS при ее запуске и выполнении файлов config.nt и autoexec.nt
ENDLOCAL	Окончание использования локального окружения в пакетной программе

Таблица 3.2 (окончание)

Команда	Описание
FINDSTR	Поиск строки в файлах с использованием регулярных выражений
NTCMDPROMPT	Запуск после активации резидентной программы (TSR) или после временного выхода из приложения MS-DOS командного интерпретатора cmd.exe системы Windows NT, а не интерпретатора command.com
POPD	Восстановление имени каталога, сохраненного командой PUSHD
PUSHD	Сохранение имени текущего каталога для команды POPD и переход в другой каталог
SETLOCAL	Начало области локальных установок переменных среды
TITLE	Задание заголовка окна командной строки
&&	Команда, следующая за этим символом, будет запущена, только если команда, стоящая перед символом, завершилась успешно
||	Команда, следующая за этим символом, будет запущена, только если предыдущая команда завершилась с ошибкой
&	Разделитель команд в командной строке
()	Группировка команд
^	Управляющий символ, позволяющий вводить команду как текст
;,	Разделители параметров

Кроме стандартных команд, очень полезными для практической работ оказываются утилиты из пакета Windows NT Workstation (Server) Resourc Kit.

Замечание

Далее мы будем рассматривать только команды, которые включены ка в Windows NT Workstation Resource Kit, так и в Windows NT Server Resource Ki и будем писать просто Windows NT Resource Kit.

Перечислим только несколько новых возможностей, которые предоставля ются этим пакетом при работе с командной строкой и пакетными файлами

- периодический запуск команды или программы через определенный ин тервал времени;
- поиск строк текста в файлах с одновременной заменой этих строк и другие;
- копирование файлов и каталогов вместе с информацией безопасност о них;

- выделение каталогов, размер которых превышает заданное значение;
- проверка наличия свободного места на диске;
- копирование прав доступа с одного совместного ресурса на другой;
- приостановка выполнения командного файла на заданное число секунд;
- вывод текущего времени и даты вместе с нужным сообщением в стандартный выходной поток;
- вычисление времени выполнения определенной команды.

Команды Windows NT Resource Kit, реализующие эти возможности, описаны в соответствующих разделах настоящей главы.

Следует, однако, иметь в виду, что некоторые распространенные команды MS-DOS и Windows 9*x*, рассмотренные в *гл. 1* и *2*, в Windows NT исключены или заменены на другие. Скажем, вместо команды DELTREE, удаляющей каталог со всеми его подкаталогами, нужно использовать утилиту RMDIR с ключом /S. Windows NT не может быть размещена на стандартных гибких дисках емкостью 1,2 Мбайт и 1,44 Мбайт, поэтому из перечня команд исключена утилита SYS, которая копировала необходимые системные файлы на заданный диск. По этой же причине в команде FORMAT удален ключ /S. В Windows NT больше нет программы FDISK — работа с жесткими дисками осуществляется с помощью "Администратора дисков" (Disk administrator). В силу того, что система автоматически оптимизирует использование жестких дисков и обеспечивает кэширование для подсистемы MS-DOS, не используются команды DEFRAG, SCANDISC, SMARTDRV, DRVSPACE. Windows NT обеспечивает автоматический доступ к дисководам CD-ROM для подсистемы MS-DOS, поэтому убрана команда MSCDEX. Также не поддерживаются команды CHOICE (хотя эта команда включена в состав Windows NT Resource Kit), CTTY, EMM386 и некоторые другие. В Windows NT недоступна множественная конфигурация подсистемы MS-DOS, которая описывалась в *разд. "Интерактивная загрузка операционной системы" гл. 2.*

Условное выполнение команд

В *командной строке* Windows NT можно использовать специальные символы, которые позволяют вводить несколько команд одновременно и управлять работой команд в зависимости от результатов их выполнения. С помощью таких символов условной обработки можно содержание небольшого пакетного файла записать в одной строке и выполнить полученную составную команду.

Используя символ амперсанда &, можно разделить несколько утилит в одной командной строке, при этом они будут выполняться друг за другом. Например, если набрать командную строку

```
DIR & PAUSE & COPY /?
```

и нажать клавишу <Enter>, то вначале на экран будет выведено содержимое текущего каталога, а после нажатия любой клавиши — встроенная справка команды COPY.

Символ ^ позволяет использовать командные символы как текст, т. е. при этом происходит игнорирование значения специальных символов. Например, если ввести в командной строке

```
ECHO Абв & COPY /?
```

и нажать клавишу <Enter>, то произойдет выполнение подряд двух команд: ECHO Абв и COPY /?. Если же выполнить команду

```
ECHO Абв ^& COPY /?
```

то на экран будет выведено

```
Абв & COPY /?
```

В этом случае просто выполняется одна команда ECHO с соответствующими параметрами.

Условная обработка команд в Windows NT осуществляется с помощью символов && и || следующим образом. Двойной амперсанд && запускает команду, стоящую за ним в командной строке, только в том случае, если команда, стоящая перед амперсандами была выполнена успешно. Например, если в корневом каталоге диска C: есть файл plan.txt, то выполнение строки

```
TYPE C:\plan.txt && DIR
```

приведет к выводу на экран этого файла и содержимого текущего каталога. Если же файл C:\plan.txt не существует, то команда DIR выполняться не будет.

Два символа || осуществляют в командной строке обратное действие, т. е. запускают команду, стоящую за этими символами, только в том случае, если команда, идущая перед ними, не была успешно выполнена. Таким образом, если в предыдущем примере файл C:\plan.txt будет отсутствовать, то в результате выполнения строки

```
TYPE C:\plan.txt || DIR
```

на экран выведется содержимое текущего каталога.

Отметим, что условная обработка действует только на ближайшую команду, т. е. в строке

```
TYPE C:\plan.txt && DIR & COPY /?
```

команда COPY /? запустится в любом случае, независимо от результата выполнения команды TYPE C:\plan.txt.

Несколько утилит можно сгруппировать в командной строке с помощью скобок.

Рассмотрим, например, две строки:

```
TYPE C:\plan.txt && DIR & COPY /?
TYPE C:\plan.txt && (DIR & COPY /?)
```

В первой из них символ условной обработки && действует только на команду DIR, во второй — одновременно на две команды: DIR и COPY.

Запуск программ в определенное время

В Windows NT с помощью утилиты AT можно запускать команды и программы в заданное время. Для работы этой команды должен быть запущен сервис расписаний (**Scheduler** в Windows NT Server или **Task Scheduler** в Windows NT Workstation), а пользователь должен являться членом локальной группы администраторов. Для того чтобы внести новое задание в расписание, используется следующий синтаксис команды AT:

```
AT [\\имя_компьютера] время [/INTERACTIVE]
[/EVERY:дата[,...]|/NEXT:дата[,...]] "команда"
```

Если запустить AT без параметров, то на экран будет выведен список всех команд и программ, которые будут запущены с ее помощью. Например:

```
Статус  Код      Дата              Время        Командная строка
-------------------------------------------------------------------
        1    Следующий Пн          10:00 AM     d:\copy1.bat
        2    Каждый Пн Вт          10:00 AM     d:\copy2.bat
```

Параметр *\\имя_компьютера* задает удаленный компьютер, на котором могут быть запущены планируемые команды и программы. Если этот параметр не задан, то предполагается, что запуск программ будет произведен на том же компьютере, где запущена команда AT.

Параметр *время* задает время, когда планируемая команда должна быть запущена. Время задается в 24-часовом формате часы:минуты (от 00:00 до 23:59).

Ключ /INTERACTIVE позволяет команде AT обмениваться данными с теми пользователями, которые будут подключены к системе в момент выполнения запланированной команды (в случае выполнения утилиты командной строки или пакетного файла будет создано новое командное окно).

Ключи /EVERY:*дата*[,...] или /NEXT:*дата*[,...] задают дату, когда должна быть запущена запланированная команда. Если указан ключ /EVERY:*дата*[,...], то команда будет запускаться в заданные дни в течение недели или месяца (например, каждый четверг или каждый третий день месяца). Дни недели задаются буквами (Пн, Вт, Ср, Чт, Пт, Сб, Вс или M, T, W, Th, F, S, Su, в зависимости от того, какая версия Windows NT (русифицированная или англий-

ская) установлена на компьютере), а дни месяца — цифрами (от 1 до 31). Несколько дат разделяются запятыми. Если параметр *дата* не задан, то подразумевается текущий день месяца.

Ключ /NEXT:*дата*[,...] позволяет запустить команду при наступлении следующей заданной даты (например, в следующий четверг). Параметр *дата* имеет то же значение, что и в ключе /EVERY.

При помощи параметра "*команда*" задаются утилита, программа (файлы с расширением exe или com) или пакетный файл (файлы с расширением bat или cmd), которые должны быть запущены. Если для задания команды необходимо указывать ее местоположение, имя файла задается полностью, с указанием пути и диска. Если команда запускается на удаленной машине, то необходимо указать имя этой машины и имя разделяемого ресурса, а не имя сетевого диска. Параметр "*команда*" должен быть заключен в кавычки.

Внимание!

Команда AT не вызывает автоматически командный интерпретатор cmd.exe перед запуском запланированной команды. Поэтому если запускается внутренняя команда (например, COPY или DIR), а не исполняемый файл, то вначале должен быть запущен командный интерпретатор с ключом /C, например, AT 23:00 "CMD /C DIR > C:\test.out".

После того как задание запланировано на выполнение, ему присваивается идентификационный номер. Отменить выполнение уже запланированного задания можно с помощью следующего варианта команды AT:

AT [*имя_компьютера*] [[*код*][/DELETE[/YES]]

Здесь параметр *код* определяет идентификационный номер, присваиваемый команде или программе, которая будет запущена. Если *код* не задан, отменены будут все запланированные на компьютере команды.

Ключ /YES задает утвердительный ответ на все запросы об отмене запланированных для запуска команд.

Замечание

Команды, запуск которых задан с помощью AT, выполняются как фоновые (если только не указан ключ /INTERACTIVE), поэтому результаты их работы не выводятся на экран. Для направления вывода результатов в файл используются, как обычно, символы перенаправления > и >>. В этом случае сама команда должна быть заключена в кавычки.

Текущим каталогом для выполнения запланированных команд по умолчанию является каталог %SystemRoot%. Все запланированные с помощью AT команды сохраняются в системном реестре, и, следовательно, не будут потеряны при перезапуске службы расписаний или всего компьютера.

Еще раз повторим, что запланированные задания, использующие сетевые диски, не должны использовать переназначения дисков, заданные пользователем, т. к. служба расписаний может не получить доступа к таким дискам или диски могут оказаться не подключенными, если другой пользователь войдет в систему в момент выполнения запланированной команды. Вместо этого, запланированные задания должны использовать полный сетевой путь (*\\имя_компьютера\имя_ресурса*).

Например, пусть у пользователя имеется сетевой диск X:, подключенный к сетевому ресурсу \\Server1\ForBackup, на который необходимо производить резервное копирование. При этом вы написали универсальный командный файл mybackup.bat, который выполняет копирование по указанному в качестве параметра командной строки пути. Чтобы запланировать запуск этого командного файла в 1 час ночи, можно использовать следующую команду:

```
AT 1:00 mybackup \\Server1\ForBackup,
```

тогда как недопустимой является следующая форма:

```
AT 1:00 mybackup x:.
```

Если с помощью AT запланирована команда, использующая буквенное обозначение диска для подключения к разделяемому каталогу, то после ее выполнения должна быть запланирована команда, отключающая данный каталог от диска. В противном случае, буква, использованная для обозначения подключаемого диска, будет недоступна или не будет выводиться в командной строке.

Приведем примеры использования команды AT.

1. Выведем список команд, которые будут выполнены на компьютере \\Server1:

   ```
   AT \\Server1
   ```

2. Отменим вызов всех команд, запланированных на данном сервере:

   ```
   AT /DELETE
   ```

3. Требуется организовать ежедневное резервное копирование на данный компьютер в каталог D:\BACKUP с сервера \\Server1 (сетевой путь \\Server1\Programs\Program1) с подключением сетевого диска. Копирование должно начинаться в 2 часа ночи, причем для уменьшения сетевого трафика копироваться должны только измененные за прошедший день файлы, включая скрытые. Для решения этой задачи можно создать файл mycopy.bat следующего содержания:

   ```
   REM Подключаем сетевой диск
   NET USE Y: \\Server1\Programs
   REM Копируем нужные данные
   XCOPY /E /D /H /C Y:\PROGRAM1 D:\BACKUP\PROGRAM1
   ```

```
REM Отключаем сетевой диск
NET USE Y: /DELETE
```

Запланировать ежедневный запуск этого файла в заданное время можно при помощи следующей команды:

```
AT 2:00 /EVERY: M,T,W,Th,F,S,Su D:\mycopy.bat
```

Другой утилитой, позволяющей планировать запуск команд или программ на локальном или сетевом компьютере, является SOON из пакета Windows NT Resource Kit. С помощью этой команды можно запустить заданную программу через небольшой интервал времени от момента планирования. На самом деле, для этой цели генерируется и выполняется соответствующая команда AT, однако SOON более удобна для использования, т. к. не нужно задавать абсолютное время — команда SOON при своем запуске планирует запуск программы или команды через определенное количество секунд от текущего момента времени.

Кроме того, с помощью SOON можно запланировать циклическое выполнение каких-либо задач с интервалом, меньшим одного дня. Для этого нужно собрать необходимые команды в один файл, включить в него команду SOON и запустить этот файл. Команда SOON нужна для того, чтобы через определенный промежуток времени вновь запустить этот файл.

Аналогично команде AT, для использования SOON на компьютере должен быть запущен сервис расписаний (**Scheduler** для Windows NT Server или **Task Scheduler** для Windows NT Workstation).

Команда SOON может использоваться в двух режимах: планирования и конфигурации.

Синтаксис для режима планирования имеет вид:

```
SOON [\\компьютер] [задержка] [/INTERACTIVE] "команда"
```

Параметр *\\компьютер* задает сетевое имя компьютера, на котором планируется запуск заданной команды (параметр *команда*). Если этот параметр опущен, то команда будет запускаться на локальном компьютере.

Параметр *задержка* определяет интервал времени (в секундах), через который будет запущена заданная команда. В случае, когда этот параметр не указан, SOON будет использовать одну из своих настроек по умолчанию (эти установки можно изменить, запустив команду SOON в режиме конфигурации, что описано ниже). При этом, если задание будет запущено на локальном компьютере, то SOON использует свой параметр LocalDelay, а если задание планируется запустить на сетевом компьютере, то SOON использует свой параметр RemoteDelay.

Ключ /INTERACTIVE позволяет запущенному заданию обмениваться данными с теми пользователями, которые будут подключены к системе в момент вы-

полнения запланированной команды. Если этот ключ не указан, то SOON использует текущее значение своего параметра InteractiveAlways (устанавливается с помощью ключа /I в режиме конфигурации) следующим образом: если InteractiveAlways включен, то запущенное задание взаимодействует с пользователем, если же InteractiveAlways выключен, то запущенное задание с пользователем не взаимодействует.

Синтаксис для режима конфигурации имеет вид:

```
SOON /D [/L:n][/R:n][/I:{ON|OFF}]
```

Ключ /D здесь указывает на то, что команда SOON должна вывести или модифицировать свои установки по умолчанию. Если этот ключ опущен, то команда запускается в режиме планирования. Команда

```
SOON /D
```

выведет на экран текущие установки, заданные по умолчанию. Если кроме ключа /D указаны другие ключи, то команда SOON изменит соответствующие параметры, установленные по умолчанию.

Ключ /L:n устанавливает значение параметра LocalDelay — величина задержки по времени, установленная по умолчанию для заданий, запускаемых на локальном компьютере. Параметр n должен быть целым положительным числом (интервал в секундах). Первоначально значение параметра LocalDelay равно 5 секундам.

Ключ /R:n устанавливает значение параметра RemoteDelay — величина задержки по времени, установленная по умолчанию для заданий, запускаемых на сетевом компьютере. Первоначальное значение RemoteDelay равно 15 секундам.

Ключ /I:[ON|OFF] устанавливает значение параметра InteractiveAlways. Первоначально этот параметр выключен (OFF).

Как уже отмечалось выше, команда SOON бывает очень полезна при создании и отладке планируемых заданий (например, пакетных файлов). При такой отладке часто требуется посмотреть, каким образом будет вести себя планируемое задание при запуске с помощью службы расписаний, в случае необходимости внести в командный файл нужные изменения и вновь запустить откорректированное задание. Непосредственное использование для этой цели команды AT может занять много времени, т. к. в AT можно задавать только абсолютное время запуска запланированного задания. Поэтому повторное планирование запуска заданий с помощью AT требует, по крайней мере, изменения времени запуска. Повторное же планирование таких заданий с помощью команды SOON может сократить время отладки.

В заключение рассмотрим следующий пример, демонстрирующий, как можно с помощью команды SOON каждые пять минут запускать определенную

программу (например, someprog.exe). С помощью команды SOON 300 someprog.exe можно запланировать лишь единственный запуск someprog.exe через 5 минут (300 секунд), поэтому для решения поставленной задачи создадим командный файл every5.bat следующего содержания:

```
SOON 300 every5.bat
someprog.exe
```

После этого с помощью AT или SOON запланируем запуск файла every5.bat (например, выполнив команду SOON every5.bat). Когда every5.bat запустится, команда SOON, стоящая в первой строке, запланирует еще один запуск данного пакетного файла через 5 минут. После этого выполнится программа someprog.exe.

Изменения в командах для работы с файловой системой

Операционная система Windows NT поддерживает две основные файловые системы: FAT, которая используется в MS-DOS и Windows 9*x*, и NTFS (NT File System, файловая система NT), которая содержит ряд изменений и усовершенствований, характерных для Windows NT. В этом разделе будут описаны как команды, работающие в обеих этих файловых системах, так и команды, специфические для NTFS.

Рассмотрим, какие изменения произошли в командах, которые уже описывались в *разд. "Работа с файловой системой" гл. 1* (CD, MD, RD, DEL, XCOPY и DIR), а также две команды MS-DOS (REPLACE и TREE), которых не было в Windows 9*x*. Кроме этого, будут описаны команда CACLS и несколько утилит командной строки из пакета Windows NT Resource Kit (SCOPY, DIRUSE, SHOWDISK и PERMS).

Команда *CD*

Начнем с команды CD, которая выводит имя текущего каталога либо производит его смену. В Windows NT в эту команду добавлен ключ /D:

```
CD [/D] [диск:][путь]
```

Этот ключ используется для одновременной смены текущих диска и каталога. Например, если текущим каталогом у вас является C:\PROGRAMS, то для того, чтобы сделать текущим каталог D:\TEXTS, достаточно ввести одну команду

```
CD /D D:\TEXTS
```

а не две

```
D:\
CD TEXTS
```

как это было в MS-DOS и Windows 9*x*.

Кроме того, при включении расширенной обработки команд (CMD /X) CD перестает рассматривать пробелы как разделители, что позволяет перейти в подкаталог, имя которого содержит пробелы, не заключая все имя каталога в кавычки. Например, команда

```
CD \Мои документы
```

приводит к тому же результату, что и команда

```
CD "\Мои документы"
```

При отключении расширенной обработки команд необходимо имя каталога, содержащего пробелы, заключать в кавычки.

Команды *MKDIR* и *RMDIR*

Изменения произошли и в командах создания нового каталога (MKDIR или MD) и удаления существующего (RMDIR или RD). При включении расширенной обработки команд MKDIR создает при необходимости не только основной каталог, но и все промежуточные каталоги в указанном пути. Например, если каталоги, заданные в качестве параметров команды MKDIR не существуют, то выполнение команды

```
MKDIR \A\B\C\D
```

приводит к тому же результату, что и выполнение следующих семи команд подряд:

```
MKDIR \A
CDR \A
MKDIR B
CD B
MKDIR C
CD C
MKDIR D
```

При отключении расширенной обработки команд для создания промежуточных каталогов необходимо будет пользоваться системой команд MKDIR и CD, приведенной во втором варианте данного примера.

Команда удаления каталогов RMDIR в Windows NT была изменена следующим образом. В эту команду добавлен ключ /S, позволяющий удалять все файлы и подкаталоги заданного каталога (удаляемые файлы не должны быть скрытыми или системными). Таким образом, команда RMDIR /S заменяет DELTREE из Windows 9*x* (команды DELTREE в Windows NT нет).

Команда *DEL*

Много новых ключей появилось и в команде DEL (напомним, что в Windows 9*x* у этой команды имелся единственный ключ /P, с помощью которого можно было выводить подтверждение для каждого удаляемого файла). В Windows NT команда DEL имеет следующий синтаксис:

```
DEL [диск:][путь]имя_файла [ ...] [/P][/F][/S][/Q][/A[:атрибуты]]
```

Опишем новые ключи этой команды.

С помощью ключа /F можно удалять файл с атрибутом "Только для чтения".

Ключ /S означает, как обычно, что нужно удалять файлы не только в заданном каталоге, но и во всех его подкаталогах. Если используется расширенный режим командного интерпретатора, то использование ключа /S позволяет осуществить при удалении вывод на экран имени каждого удаляемого файла.

Ключ /Q задает режим работы без подтверждения удаления.

Использование ключа /A позволяет удалить файлы с заданными атрибутами. Параметр *атрибуты* может иметь следующие значения: R — "Только для чтения", H — "Скрытый", S — "Системный", A — "Архивный". Например, чтобы удалить все скрытые файлы в текущем каталоге, нужно выполнить следующую команду:

```
DEL *.* /A:H
```

Также использование ключа /A позволяет удалять только те файлы, которые не имеют заданных атрибутов. Для этого в параметре *атрибуты* нужно указать знак - (минус). Например, чтобы удалить все файлы в текущем каталоге, которые не являются скрытыми, надо выполнить следующую команду:

```
DEL *.* /A:-H
```

Команды копирования

Перейдем теперь к рассмотрению команд, связанных с копированием файлов и каталогов.

В Windows NT вновь включена отсутствовавшая в Windows 9*x* команда MS-DOS REPLACE, которая позволяет производить замену файлов в одном каталоге файлами с теми же именами из другого каталога. Также команда REPLACE может быть использована для добавления только тех файлов, которых еще нет в каталоге. Синтаксис этой команды имеет вид:

```
REPLACE [диск1:][путь1]имя_файла [диск2:][путь2] [/A][/P][/R][/W]
REPLACE [диск1:][путь1]имя_файла [диск2:][путь2] [/P][/R][/S][/W][/U]
```

Здесь параметр [*диск1*:][*путь1*]*имя_файла* задает местонахождение и имя файла или набора файлов, предназначенных для копирования, а параметр

[*диск2*:][*путь2*] — местонахождение файла-результата. Имена замещаемых файлов задавать нельзя. Если диск и каталог, являющиеся источником, не заданы, то команда REPLACE использует к качестве целевого текущий диск и текущий каталог.

Использование ключа /A позволяет добавлять в каталог-результат только новые файлы из каталога-источника (без перезаписи уже имеющихся). Этот ключ нельзя использовать с ключами /S и /U.

И при замене, и при добавлении файлов команда REPLACE выводит их имена на экран. По окончании работы команды на экран выводится итоговая строка в одном из следующих форматов:

```
nnn файлов добавлено nnn файлов заменено
ни один файл не добавлен ни один файл не заменен
```

При использовании ключа /P на экран перед заменой целевого или добавлением исходного файла выводится запрос на подтверждение.

Команда REPLACE может производить замещение не только обычных файлов, но и файлов, защищенных от записи. Это достигается использованием в командной строке ключа /R. Если этот ключ не задан, но программа пытается заменить файл, защищенный от записи, то на экран будет выведено сообщение об ошибке и операция перемещения будет остановлена.

Использование ключа /S позволяет производить поиск по всем подкаталогам целевого каталога и заменять файлы с совпадающими именами. Этот ключ нельзя использовать совместно с ключом /A. Поиск в каталогах, заданных параметром *путь1*, не производится.

Если в команде REPLACE используется ключ /W, то перед началом поиска исходных файлов система будет ждать, пока пользователь вставит диск в дисковод. Если ключ /W не задан, замена или добавление файлов начнется сразу же после нажатия клавиши <Enter>.

Использование ключа /U задает режим замены (обновления) только тех файлов, которые имеют более раннюю дату модификации, чем файлы в исходном каталоге. Этот ключ не может быть использован совместно с ключом /A.

Замечание

Команда REPLACE не может быть использована для обновления скрытых или системных файлов. Чтобы работать с такими файлами, необходимо предварительно сменить их атрибуты с помощью команды ATTRIB.

В табл. 3.3 приведены коды завершения команды REPLACE с их кратким описанием.

Таблица 3.3. *Коды завершения команды* REPLACE

Код	Описание
0	Файлы успешно заменены или добавлены командой REPLACE
1	Команда REPLACE обнаружила некорректную версию MS-DOS
2	Команда REPLACE не может найти исходные файлы
3	Команда REPLACE не может найти путь источника или результата
5	Пользователь не имеет доступа к заменяемым файлам
8	Для работы команды недостаточно системной памяти
11	В строке вызова команды содержится синтаксическая ошибка

Приведем несколько примеров использования команды REPLACE.

1. Пусть несколько каталогов на диске C: содержат различные версии файла phones.cli, который содержит имена клиентов и их телефонные номера. Для замены всех этих файлов новейшей версией с диска A: служит команда:

```
REPLACE A:\phones.cli C:\ /S
```

2. Предположим, требуется добавить новый драйвер принтера в каталог TOOLS на диске C:, в котором уже содержится несколько файлов драйверов принтеров для текстовых процессоров. Для этого можно использовать следующую команду:

```
REPLACE A:*.prd C:\TOOLS /A
```

   Эта команда просматривает текущий каталог диска A:, находит все файлы с расширением prd, а затем добавляет эти файлы в каталог TOOLS на диске C:. Так как задан ключ /A, будут добавлены только те файлы, которых еще нет на диске C:.

Несколько новых ключей появилось в часто используемой команде XCOPY. В Windows NT эта команда имеет следующий синтаксис:

```
XCOPY источник [результат] [/W][/P][/C][/V][/Q][/F][/L][/D[:дата]][/U]
[/I][/S][/E]][/T][/K][/R][/H][/A|/M][/N][/EXCLUDE:имя_файла][/Z]
```

Напомним, что если в XCOPY не задан параметр *результат*, то файлы будут копироваться в текущий каталог. Опишем новые и измененные ключи для команды XCOPY.

Ключ /V, обеспечивающий проверку каждого скопированного файла на соответствие его оригиналу, теперь игнорируется, т. к. такая проверка обеспечивается самой операционной системой Windows NT. Данный ключ предназначен для обеспечения совместимости с предыдущими версиями MS-DOS.

Из операции копирования можно исключить один или несколько файлов. Это делается с помощью ключа /EXCLUDE:*имя_файла*. Здесь в файле, заданном параметром *имя_файла*, можно задать список исключений для копирования. В этом списке нужно указывать по одному имени в строке, но без использования символов подстановки. Рассмотрим пример. Пусть в текущем каталоге C:\ADITOR записаны три файла с расширением txt:

```
readme.txt
letter1.txt
letter2.txt
```

Создадим файл excl.txt и запишем в него две строки:

```
readme.txt
excl.txt
```

Если теперь выполнить команду

```
XCOPY *.txt D:\TEXT /EXCLUDE:excl.txt
```

то в каталог D:\TEXT будут скопированы только два файла: letter1.txt и letter2.txt.

Замечание

Если заданное исключение совпадает с частью полного имени копируемого файла, то этот файл копироваться не будет.

Наконец, использование ключа /Z осуществляет копирование по сети в режиме перезапуска.

Напомним, что в файловой системе NTFS у каждого файла/каталога имеется владелец (обычно это пользователь, который его создал), а также может вестись аудит для этого файла/каталога. При копировании файлов или каталогов с помощью команд COPY и XCOPY владельцем копии становится пользователь, выполнявший копирование, а информация об аудите для копии пропадает. Иногда бывает необходимо оставить прежнего владельца для файла-копии и сохранить информацию об аудите доступа. Такие возможности предоставляет команда SCOPY из пакета Windows NT Resource Kit, которая копирует файлы и каталоги в разделе NTFS вместе с установками безопасности для них. Синтаксис этой команды имеет вид:

```
SCOPY источник [результат] [/O][/A][/S]
```

Использование ключа /O позволяет копировать информацию о владельце копируемого файла или каталога.

При использовании ключа /A происходит копирование информации, связанной с аудитом доступа к файлу или каталогу.

Использование ключа /S позволяет копировать файлы из всех подкаталогов указанного каталога.

Для копирования файлов, владельцем которых вы являетесь, вам не нужно никаких специальных пользовательских прав. Однако для использования ключей /O или /A, или для копирования файлов других пользователей, к которым в обычном режиме вы не имеете доступа, вы должны входить в группу администраторов и на том компьютере, откуда производится копирование, и на том, куда копируются данные.

В ряде случаев команда SCOPY пытается использовать специальные привилегии пользователей. Если вам разрешено архивировать файлы и каталоги ("Backup files and folders"), то вы можете копировать те файлы, к которым в обычном режиме у вас нет прав доступа. Эта же привилегия требуется в том случае, когда вы используете ключ /O при копировании тех файлов, владельцем которых вы не являетесь. Для использования ключа /A у вас должна иметься привилегия на управление аудитом и журналом безопасности ("Manage auditing and the security log").

Команда *DIR*

Перейдем теперь к рассмотрению команды DIR, которая, как мы знаем, позволяет получить в различных форматах список файлов и подкаталогов какого-либо каталога. В Windows NT в DIR произошли изменения ключей, и теперь синтаксис этой команды имеет вид:

```
DIR [диск:] [путь] [имя_файла] [;...] [/P][/W][/D][/A[[:]атрибуты]]
[/O[[:]порядок]][/T[[:]дата]][/S][/B][/L][/N][/X]
```

В команде DIR может быть задано несколько файлов, информацию о которых необходимо вывести. В этом случае имена файлов могут быть разделены пробелами, запятыми или точкой с запятой. Например, если, находясь в каталоге C:\ADITOR, задать команду

```
DIR aditor.exe;aditor.hlp
```

то на экран выведется следующая информация о двух этих файлах:

```
Том в устройстве C имеет метку WinNT
 Серийный номер тома: 646D-ECBA
 Содержимое каталога C:\aditor
01.12.99  22:13               461 312 aditor.exe
 Содержимое каталога C:\aditor
08.10.98  22:12                24 594 aditor.hlp
           2 файл(а,ов)        485 906 байт
                           796 328 960 байт свободно
```

Рассмотрим, какие возможности предоставляют новые ключи команды DIR.

Напомним, что при использовании ключа /W вывод информации производится в несколько колонок с максимально возможным числом имен файлов

или каталогов на одной строке. Применение ключа /D позволяет делать то же самое, но с дополнительной сортировкой по колонкам, например:

```
Том в устройстве C имеет метку WinNT
 Серийный номер тома: 646D-ECBA
 Содержимое каталога C:\aditor
[.]             aditor.hlp      templt01.dat    UNINST0.000
[..]            hilite.dat      templt02.dat    UNINST1.000
aditor.exe      readme.txt      ttable.dat
          11 файл(а,ов)          533 647 байт
                              796 323 840 байт свободно
```

Параметр *дата* в ключе /T задает, какое время используется при выводе и сортировке информации о файлах и каталогах. В этом параметре можно задавать следующие значения: C — дата создания (Creation), A — дата последнего доступа (Access), W — дата последней записи (Write).

Использование ключа /N позволяет выводить список файлов в расширенном формате, с расширениями в правой колонке, например:

```
Том в устройстве C имеет метку WinNT
 Серийный номер тома: 646D-ECBA
 Содержимое каталога C:\aditor
21.03.01  17:06         <КАТАЛОГ>       .
21.03.01  17:06         <КАТАЛОГ>       ..
01.12.99  22:13                461 312 aditor.exe
08.10.98  22:12                 24 594 aditor.hlp
18.09.98  17:55                  1 082 hilite.dat
25.01.00  16:26                  3 974 readme.txt
07.08.98  00:00                     48 templt01.dat
07.08.98  00:00                    227 templt02.dat
07.08.98  00:00                    357 ttable.dat
15.04.98  01:08                 40 960 UNINST0.000
02.03.99  07:36                  1 093 UNINST1.000
          11 файл(а,ов)          533 647 байт
                              796 323 840 байт свободно
```

Применение ключа /X задает режим вывода сокращенных имен для файлов систем NTFS и FAT. Формат вывода команды DIR с ключом /X совпадает с форматом вывода при использовании ключа /N, но сокращенные имена выводятся после полных, например:

```
Том в устройстве C имеет метку WinNT
 Серийный номер тома: 646D-ECBA
 Содержимое каталога C:\aditor
21.03.01  17:09         <КАТАЛОГ>                       .
21.03.01  17:09         <КАТАЛОГ>                       ..
```

```
01.12.99  22:13           461 312                  aditor.exe
08.10.98  22:12            24 594                  aditor.hlp
18.09.98  17:55             1 082                  hilite.dat
25.01.00  16:26             3 974                  readme.txt
07.08.98  00:00                48                  templt01.dat
07.08.98  00:00               227                  templt02.dat
07.08.98  00:00               357                  ttable.dat
15.04.98  01:08            40 960                  UNINST0.000
02.03.99  07:36             1 093                  UNINST1.000
21.03.01  17:09                27 ВТОРОЙ~1.TXT     Второй файл.txt
21.03.01  17:09                27 ПЕРВЫЙ~1.TXT     Первый файл.txt
          13 файл(а,ов)          533 701 байт
                              796 323 840 байт свободно
```

Заметим также, что теперь в команде DIR отсутствуют ключи /V и /4, которые были у нее в Windows 9*x*.

Команда *TREE*

Другой командой, предназначенной для вывода информации о файлах и каталогах, является TREE (эта команда не была включена в Windows 9*x*). С ее помощью можно получить графическое представление дерева каталогов заданного пути или диска. Синтакис этой команды имеет вид:

```
TREE [диск:][путь] [/F][/A]
```

Параметр *диск*: задает диск, графическое представление дерева каталогов которого будет выведено, параметр *путь* указывает каталог, для которого будет выведена структура подкаталогов (рис. 3.2).

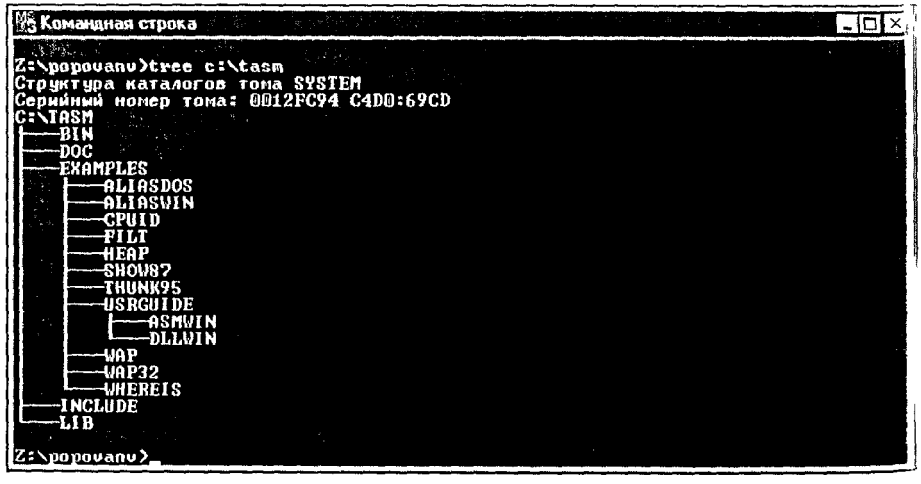


Рис. 3.2. Структура каталогов, выводимая командой TREE

Использование ключа /F позволяет выводить имена файлов в каждом каталоге (рис. 3.3).

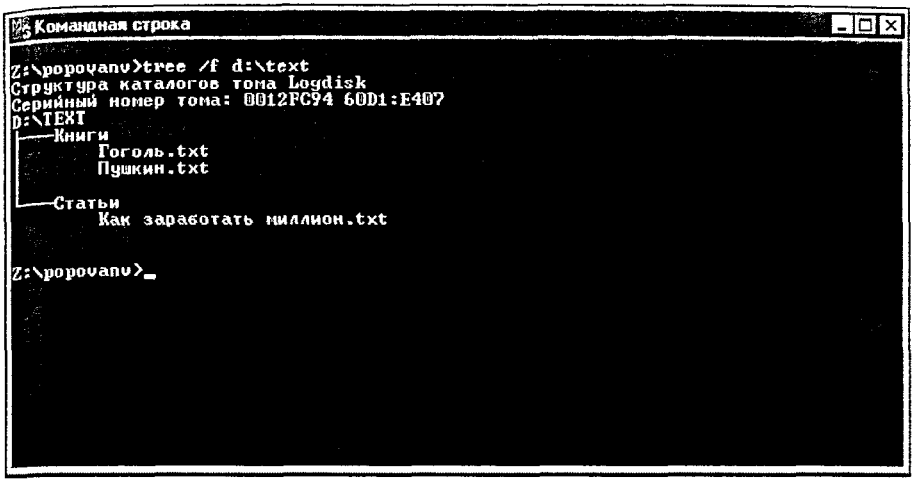


Рис. 3.3. Структура каталогов с указанием хранящихся в них файлов

Применение ключа /A позволяет использовать текстовые символы вместо графических для вывода связей между каталогами.

Команды *DIRUSE* и *SHOWDISK*

Рассмотрим теперь две полезные утилиты из пакета Windows NT Resource Kit, предназначенные для работы с файловой системой.

Очень удобной является команда DIRUSE, которая показывает, сколько места на диске занимают определенные каталоги. С помощью этой утилиты вы можете, например, быстро проверить, какой объем занимают домашние каталоги пользователей. Кроме того, DIRUSE может выделить все каталоги, размер которых превышает заданное вами значение. Это очень удобно, т. к. в Windows NT нет встроенного механизма квотирования, т. е. ограничения дискового пространства, выделяемого пользователю (в Windows 2000 квотирование имеется).

Замечание

Если вы являетесь членом группы администраторов, то с помощью команды DIRUSE вы сможете проверить объем даже тех каталогов в разделе NTFS, на которые у вас нет прав доступа.

Синтаксис команды DIRUSE имеет вид:

```
DIRUSE {/S|/V}{/M|/K|/B}[/C][/,][/Q:#[/L][/A][/D][/O]][/*] каталоги
```

Обязательный параметр *каталоги* задает список каталогов для проверки (названия каталогов разделяются пробелами).

Ключи для команды DIRUSE могут быть заданы в произвольном порядке, вместо символов / можно указывать -.

Использование ключа /S позволяет выводить информацию обо всех подкаталогах заданного каталога.

Применение ключа /V выводит сообщения о процессе выполнения команды во время сканирования подкаталогов. Этот ключ игнорируется, если указан /S.

Использование следующих ключей определяет, в каких единицах будет выводиться объем проверяемых каталогов. Если указан ключ /M, то размеры печатаются в мегабайтах, если ключ /K — в килобайтах, если ключ /B — то в байтах (этот режим принимается по умолчанию). Например, для того, чтобы посчитать размер в мегабайтах каталога C:\DOCUM и число файлов в нем, нужно задать такую команду:

```
DIRUSE /M C:\DOCUM
```

На экран выведется информация следующего вида:

```
Size (mb)  Files  Directory
    41.47     34  SUB-TOTAL: C:\DOCUM
    41.47     34  TOTAL: C:\DOCUM
```

Если указан ключ /C, то при подсчете будет учитываться размер не исходного, а сжатого файла.

С помощью ключа /, в выводимых на экран размерах файлов будут подставляться разделители разрядов (запятые или точки).

Если указан ключ /Q:#, то имена каталогов, размер которых превышает заданное параметром # значение, при выводе будут отмечаться знаком !. В случае, когда не указаны ключи /M (мегабайты) или /K (килобайты), то предполагается, что размер (#) задается в байтах. Например, для того чтобы выявить все подкаталоги каталога C:\DOCUM, размер которых превышает 5 Мбайт, нужно задать такую команду:

```
DIRUSE /S /M /Q:5 C:\DOCUM
```

При этом на экран выводится информация следующего типа:

```
Searching for directories that have exceeded 5 megabytes
    Size (mb)  Files  Directory
!       17.06      1  C:\DOCUM
         0.00      0  C:\DOCUM\Books
!       20.20      4  C:\DOCUM\Books\Text
         1.51     19  C:\DOCUM\ODBC
```

```
          2.69      10  C:\DOCUM\ODBC\SYMBOLS
!        41.47      34  SUB-TOTAL: C:\DOCUM

!        41.47      34  TOTAL: C:\DOCUM
```

Если задан ключ /Q, то в случае превышения хотя бы одним каталогом указанного размера, команда DIRUSE возвратит код выхода (ERRORLEVEL), равный единице. В противном случае код выхода будет равен нулю. Это свойство можно использовать в командных файлах.

Использование ключа /L позволяет записывать имена каталогов, размеры которых превышают указанный предел. Запись производится в файл diruse.log в текущем каталоге.

Если указан ключ /A, то в случае превышения каким-либо каталогом размера, указанного в ключе /Q:#, будет генерироваться сообщение (Alert). Для того чтобы этот ключ сработал, необходимо предварительно запустить службу сообщений (**Alerter**).

При использовании ключа /D выводится информация только о тех каталогах, размер которых превышает заданное значение. Например, задав команду:

```
DIRUSE /S /M /Q:5 /D C:\DOCUM
```

на экране мы увидим следующее:

```
Searching for directories that have exceeded 5 megabytes
    Size (mb)  Files  Directory
    Not Found      0  DIRUSE
!       17.06      1  C:\DOCUM
!       20.20      4  C:\DOCUM\Books\Text
!       41.47     34  SUB-TOTAL: C:\DOCUM

        41.47     34  TOTAL
```

Использование ключа /O позволяет не отмечать при просмотре подкаталоги, размер которых превышает заданное значение.

Применение ключа /* позволяет выводить информацию с группировкой по подкаталогам верхнего уровня. Например, задав команду

```
DIRUSE /S /M /Q:5 /* C:\DOCUM
```

мы получим следующие сообщения:

```
Searching for directories that have exceeded 5 megabytes
    Size (mb)  Files  Directory
         0.00      0  C:\DOCUM\Books
!       20.20      4  C:\DOCUM\Books\Text
!       20.20      4  SUB-TOTAL: C:\DOCUM\Books
```

```
  1.51      19  C:\DOCUM\ODBC
  2.69      10  C:\DOCUM\ODBC\SYMBOLS
  4.20      29  SUB-TOTAL: C:\DOCUM\ODBC

 24.40      33  TOTAL
```

В пакете Windows NT Resource Kit имеется команда SHOWDISK, с помощью которой можно считать информацию из ключа

```
HKEY_LOCAL_MACHINE \ SYSTEM\ DISK
```

системного реестра и отобразить ее на экране Данный ключ содержит информацию о каждом первичном разделе жесткого диска и обо всех логических устройствах, определенных на компьютере Команда SHOWDISK запускается без дополнительных параметров и выводит информацию следующего типа:

```
Opening \SYSTEM\DISK successful
Disk Registry Information Size..... 168
Operating System Version..... 3
Checksum..... 0x140300
Dirty Shutdown?..... 114
<reserved 1>..... 0x0
<reserved 2>..... 0x6d
<reserved 3>..... 0x0
Disk Info Offset..... 0x2c

Disk Info Size..... 124
FT Info Offset..... 0xa8
FT Info Size..... 0
FT Stripe Width..... -553123840
FT Pool Size..... 1311456
Name Offset <not implemented>..... 0x1402e0
Name Size <not implemented>..... 5505107
General Disk Information:
Number of Disks..... 2
<reserved>..... 0

Disk #0:
Number Of Partitions..... 1
<reserved>..... 0x0
Signature..... 0x351d0ce2

Partition #1:
FT Type..... Not a Fault Tolerance Partition
FT State..... Healthy
Starting Offset..... 0x4600
Length..... 212011520
```

```
FtLength.............. 0
<reserved 1>.......... 0x0
<reserved 2>.......... 0x0
Drive Letter..........
Assign Drive Letter?.. Yes
Logical Number........ 1

Ft Group.............. Not an FT Partition
Modified?............. Yes
<reserved char 0>..... 0x0
<reserved char 1>..... 0x0
<reserved char 2>..... 0x0
```

Файловая система NTFS поддерживает объектную модель безопасности Windows NT и рассматривает все тома, каталоги и файлы как самостоятельные объекты. Каждый раз, когда пользователь обращается к объекту файловой системы, его права проверяются по таблице контроля доступа (ACL, Access Control List) данного объекта. Если пользователь обладает достаточным уровнем прав, то его запрос удовлетворяется; в противном случае запрос отклоняется

Замечание

Такая модель безопасности применяется как при локальной регистрации пользователей на компьютерах с Windows NT, так и при удаленных сетевых запросах

Команда *CACLS*

Для просмотра или изменения таблиц контроля доступа файлов из командной строки предназначена стандартная команда CACLS.

```
CACLS имя_файла [/T][/E][/C][/G доступ][/R пользователь [...]]
[/P доступ [...]][/D пользователь [...]]
```

Если запустить эту команду без ключей, то на экран выведется информация о правах доступа различных пользователей и групп к заданному файлу или каталогу, т. е. таблица контроля доступа.

Использование ключей /T и /E определяет тип действия, которое будет совершено с ACL заданного файла или каталога. Если задан ключ /T, то имеющаяся ACL будет полностью заменена на новую, структура которой определяется ключами /G, /R, /P и /D Если же задан ключ /E, то в имеющуюся ACL будут просто внесены необходимые изменения (без ее полной замены)

Применение ключа /C означает, что в ACL будут вноситься изменения даже при наличии ошибок (режим игнорирования ошибок).

Использование ключа /G *доступ* задает уровень доступа пользователя или группы пользователей. Параметр *доступ* задается в следующем виде *имя*:*код_доступа*, где *имя* определяет имя пользователя или группы пользователей, а *код_доступа* может принимать одно из трех значений: R — чтение (Read), C — изменение, запись (Change), F — полный доступ (Full control).

Применение ключа /R *пользователь* позволяет отозвать права доступа у заданного пользователя.

Если задан ключ /P *доступ*, то права доступа указанного пользователя или группы пользователей будут заменены. Как и в случае применения ключа /G, параметр *доступ* задается в виде *имя*:*код_доступа*, где *код_доступа* может принимать одно из четырех значений: N — нет доступа (None), R — чтение (Read), C — изменение, запись (Change), F — полный доступ (Full control)

Использование ключа /D *пользователь* означает запрет доступа заданного пользователя (No access).

Замечание

В команде CACLS можно использовать более одного файла или пользователя

Отметим также, что CACLS не содержит опции /Y, которая автоматически давала бы утвердительный ответ на все задаваемые вопросы. Для этой цели можно применить конвейеризацию с использованием команды ECHO:

```
ECHO Y| CACLS ...
```

Рассмотрим несколько примеров использования команды CACLS.

1. Изменим ACL для каталога C:\DOCUM таким образом, чтобы доступ к нему имели только члены группы администраторов:

   ```
   CACLS C:\DOCUM /T /G Administrators:F
   ```

2. Теперь дополнительно разрешим полный доступ к этому каталогу пользователю Kazakov:

   ```
   CACLS C:\DOCUM /E /G Kazakov:F
   ```

3. Изменим права доступа для пользователя Kazakov таким образом, чтобы он мог только просматривать содержимое C:\DOCUM:

   ```
   CACLS C:\DOCUM /E /P Kazakov:R
   ```

4. Запретим доступ пользователя Kazakov к каталогу C:\DOCUM:

   ```
   CACLS C:\DOCUM /E /D Kazakov
   ```

5. Отзовем у пользователя Kazakov права доступа к каталогу C:\DOCUM:

   ```
   CACLS C:\DOCUM /E /R Kazakov
   ```

Команда *PERMS*

Утилита PERMS из пакета Windows NT Resource Kit позволяет вывести информацию о правах доступа заданного пользователя к определенному файлу или набору файлов.

Для того чтобы использовать PERMS, вы должны иметь права "Backup files and folders" на том компьютере, где хранятся файлы. Также вы должны быть членом группы администраторов в том домене или на том компьютере, где определена учетная запись того пользователя, чьи права вас интересуют. В противном случае может возникнуть ошибка доступа ("Access denied").

Синтаксис команды PERMS имеет вид:

```
PERMS [учетная_запись] [путь] [/I][/S]
```

Параметр *учетная_запись* задает имя пользователя, права доступа которого проверяются. Имя пользователя может задаваться в трех форматах:

- домен\пользователь
- компьютер\пользователь
- пользователь

Параметр *путь* задает имя файла или каталога. Здесь можно использовать групповые символы ? и *, а также сетевые пути *\\имя_компьютера\имя_ресурса*.

Использование ключа /I указывает, что проверяемый пользователь регистрируется локально на том компьютере, который определяется параметром *путь*. Если этот ключ не задан, то подразумевается, что пользователь может регистрироваться в сети с другого компьютера.

Применение ключа /S позволяет проверить права доступа пользователя к файлам во всех подкаталогах заданного пути.

В результате работы команды PERMS выводится информация, в которой используются символы, приведенные в табл. 3.4.

***Таблица 3.4.** Символы команды PERMS*

Символ	Назначение
R	Чтение
W	Запись
X	Выполнение
D	Удаление
P	Изменение прав доступа

Таблица 3.4 (окончание)

Символ	Назначение
O	Изменение владельца файла или каталога
A	Полный доступ
-	Нет доступа
*	Проверяемый пользователь является владельцем файла или каталога
#	Группа, членом которой является пользователь, является владельцем файла или каталога
?	Права доступа пользователя не могут быть определены

Например, чтобы проверить, какие права доступа к каталогу C:\DOCUM имеются у пользователя Kazakov, нужно выполнить команду

```
PERMS Kazakov C:\Docum
```

На экран выведется информация следующего вида:

```
RWXD---  C:\Docum\
```

Поиск и замена текста в файлах

В этом разделе будут рассмотрены две утилиты, одна из которых (стандартная команда FINDSTR) позволяет производить поиск нужных строк в файлах с использованием так называемых *регулярных выражений*, а другая (команда MUNGE из пакета Windows NT Resource Kit) может заменять найденные в файлах строки на новые.

Для поиска заданных строк текста в файлах в Windows NT можно по-прежнему использовать команду FIND, рассмотренную в *гл. 1*, однако команда FINDSTR предоставляет для этого намного больше возможностей. Как и FIND, команда FINDSTR способна производить поиск заданной строки текста в одном или нескольких файлах, обеспечивая при этом дополнительные возможности поиска (эти возможности будут рассмотрены ниже при описании ключей команды). На практике, однако, часто возникают ситуации, когда имеется только часть информации, которая должна быть найдена, или требуется найти информацию в широком диапазоне (например, все слова, начинающиеся с "Win"). В таких случаях команда FINDSTR предоставляет возможность поиска с использованием регулярных выражений, которые в противоположность точному заданию строки символов для поиска позволяют задать образец (шаблон) такого текста. Для задания образца используются *литералы* и *метасимволы*. Каждый символ, который не имеет специального значения в регулярных выражениях, рассматривается как литерал и

должен точно совпасть при поиске. Например, буквы и цифры являются литеральными символами. Метасимволы — это символы со специальным значением (оператор или разделитель) в регулярных выражениях. В табл. 3.5 приведены метасимволы, применяемые в команде FINDSTR.

Таблица 3.5. *Метасимволы для команды* `FINDSTR`

Символ	Значение
. (точка)	Символ подстановки: соответствует любому символу
*	Повторитель: означает ноль или более предшествующих символов или классов символов
^	Положение в строке: обозначает начало строки
$	Положение в строке: обозначает конец строки
[класс]	Класс символов: обозначает любой символ из указанного множества. Например, метасимволу [ABC] соответствуют символы A, B или C
[^класс]	Инвертированный класс: обозначает любой символ, не входящий в указанное множество. Например, метасимволу [^ABC] соответствуют любые символы, кроме A, B и C
[x-y]	Диапазон: обозначает любой символ из указанного промежутка. Например, метасимволу [A-D] соответствуют символы A, B, C или D
\x	Исключение: определяет использование метасимвола x как литерала. Например, метасимвол \$ задает символ $
\<xyz	Положение в слове: означает начало слова
xyz\>	Положение в слове: означает конец слова

Специальные символы в регулярных выражениях дают наилучший результат при совместном использовании. Например, задать любую строку символов можно с помощью комбинации символа подстановки . и повторителя *:

```
.*
```

Это выражение полезно при использовании в более сложных регулярных выражениях, таких как

```
C.*er
```

которое задает любую строку, начинающуюся с символа C и заканчивающуюся на er (cooler, caterer и т. п.) .

Синтаксис команды FINDSTR имеет вид:

```
FINDSTR [/B] [/E] [/L] [/C:строка] [/R] [/S] [/I] [/X] [/V] [/N] [/M] [/O] [/G:файл]
[/F:файл] строки файлы
```

Параметр *строки* задает текст для поиска, а параметр *файлы* определяет один или несколько файлов, в которых будет производиться поиск. В параметре *файлы* можно использовать групповые символы ? и *. Например, для поиска слова Cooler во всех файлах, название которых содержит семь символов и начинается с price, можно использовать следующую команду:

```
FINDSTR Cooler price??.*
```

Для отделения строк поиска друг от друга следует использовать пробелы, кроме тех случаев, когда задан ключ /C, задающий литеральную строку поиска. Например, команда

```
FINDSTR "red ball" letter.txt
```

ищет слова red или ball в файле letter.txt, а команда

```
FINDSTR /C:"red ball" letter.txt
```

ищет в файле letter.txt целую строку red ball.

Использование ключей /B и /E определяет, в каком месте строки следует производить сравнение. Если задан ключ /B, то сравнение производится в начале строки, если /E — то в конце строки. Например, чтобы найти строки текста, начинающиеся со слов Windows NT, во всех файлах текущего каталога с расширением txt, нужно выполнить следующую команду:

```
FINDSTR /B "Windows NT" *.txt
```

Если же требуется найти строки, которые заканчиваются этими словами, то применяется команда

```
FINDSTR /E "Windows NT" *.txt
```

Применение ключа /L означает, что заданная строка будет использоваться для поиска побуквенно.

Использование ключа /R позволяет рассматривать строку для поиска как регулярное выражение. Этот ключ не является обязательным, т. к. команда FINDSTR рассматривает метасимволы в строке для поиска как части регулярного выражения, если только не задан ключ /L.

Команда FINDSTR может производить поиск во всех подкаталогах заданного каталога. Это осуществляется, как обычно, с помощью ключа /S.

Поиск без различия строчных и заглавных букв можно задать, указав ключ /I. Например, для поиска в каждом файле текущего каталога и в файлах всех подкаталогов слова Windows без учета строчных и заглавных букв может быть использована следующая команда:

```
FINDSTR /S /I Windows *.*
```

Следующие шесть ключей отвечают за информацию, выводимую в результате поиска. Использование ключа /X указывает, что нужно выводить точно

совпавшие строки. Напротив, задав ключ /V, мы выведем строки, не содержащие совпадений. Применив ключ /N, можно напечатать в начале совпавшей строки ее номер. Ключ /M позволяет при обнаружении совпадения выводить только имя файла, а ключ /O — печатать смещение перед выводом строки с совпадением.

Если требуется найти несколько разных наборов символов в одном или нескольких файлах, то можно создать текстовый файл, каждая строка которого содержит образец для поиска. Имя такого файла нужно задать в ключе /G:*файл*.

Также можно задать точный список файлов, в которых будет производиться поиск. Этот список помещается в текстовый файл, задаваемый в ключе /F:*файл*.

Информацию, выводимую командой FINDSTR, можно перенаправить на принтер или в текстовый файл. Для этого применяются, как обычно, символы > и >>.

Рассмотрим еще несколько примеров использования команды FINDSTR.

1. Для поиска во всех файлах с программами на языке Бэйсик (расширение bas) строк текста, содержащих слово FOR, перед которым идет несколько пробелов (чтобы найти в программе оператор цикла), и для вывода номера каждой такой строки можно использовать команду

   ```
   FINDSTR /B /N /C:" *FOR" *.bas
   ```

2. Предположим, что образцы строк для поиска находятся в файле finddata.txt, а список файлов, в которых будет производиться поиск, записан в файле filelist.txt. Для организации поиска с использованием этих файлов и записи результатов работы программы в файл results.out служит команда:

   ```
   FINDSTR /G:finddata.txt /F:filelist.txt > results.out
   ```

3. Пусть необходимо найти в текущем каталоге и его подкаталогах каждый файл, содержащий слово computer без различия строчных и заглавных букв. Для вывода списка таких файлов можно использовать команду:

   ```
   FINDSTR /S /I /M "\<computer\>" *.*
   ```

   Предположим теперь, что требуется найти не только само слово computer, но и все другие слова, начинающиеся с букв co, таких как cobra или Cocker. Для этого можно использовать следующую команду:

   ```
   FINDSTR /S /I /M "\<co.*" *.*
   ```

Перейдем теперь к рассмотрению команды MUNGE из пакета Windows NT Resource Kit, с помощью которой можно выполнить операцию поиска и замены строк текста над заданным набором файлов.

Синтаксис этой команды имеет вид:

```
MUNGE файл_шаблонов [-q[-e][-o]][-r][-c][-i][-m][-z][-@][-n][{-l|-L}][-s]
[-a][-f][-t][-k][-u файл_отмены][-v] файлы_для_замены
```

Здесь параметр *файл_шаблонов* задает имя файла, в котором содержатся строки, предназначенные для поиска, и строки, использующиеся для замены. Эти строки следует располагать парами, по одной паре на строке:

```
"Старая строка" "Новая строка"
```

Также в этом файле могут быть указаны шаблоны файлов, в которых нужно выполнить поиск и замену. Для задания такого шаблона строка должна начинаться с ключа `-F` и иметь следующий вид:

```
-F {.EXT | NAME. | NAME.EXT}
```

Параметр `.EXT` задает определенное расширение, `NAME.` — определенное имя (без расширения), `NAME.EXT` — определенное имя вместе с расширением. Если вам необходимо задать несколько шаблонов файлов для обработки, их нужно располагать на разных строках.

Например, если вам нужно в файлах `probrep.txt` и `readme.txt`, находящихся в текущем каталоге, заменить все слова `CD-ROM` и `Alpha` на `hard drive` и `I386` соответственно, то содержимое файла с шаблонами (назовем его `strings.msk`) должно быть таким:

```
"CD-ROM" "hard drive"
"Alpha" "I386"
-F PROBREP.TXT
-F README.TXT
```

Если в файле с шаблонами не указан параметр `-F`, то по умолчанию обрабатываются файлы со следующими расширениями: `asm`, `c`, `cli`, `cpp`, `cxx`, `def`, `h`, `hxx`, `idl`, `inc`, `mak`, `rc`, `s`, `src`, `srv`, `tk`.

Параметр *файлы_для_замены* указывает путь к файлам, в которых производится поиск. Например, указав в качестве этого параметра точку (.), мы будем обрабатывать текущий каталог.

По умолчанию в процессе работы `MUNGE` выводятся названия файлов, в которых найдены требуемые строки, и сами строки, в которых сделаны замены. Например, в результате выполнения команды

```
MUNGE string.msk .
```

на экран выведется информация следующего вида:

```
Reading script file - 1.msk 0.tokens and 2 literal strings
Scanning .\*.*
.\PROBREP.TXT(19) : Matched "CD-ROM", replace with "hard drive"
.\PROBREP.TXT(94) : Matched "CD-ROM", replace with "hard drive"
```

```
.\PROBREP.TXT(95) : Matched "CD-ROM", replace with "hard drive"
.\PROBREP.TXT
.\README.TXT(13) : Matched "Alpha", replace with "I386"
.\README.TXT(16) : Matched "Alpha", replace with "I386"
.\README.TXT(261) : Matched "Alpha", replace with "I386"
.\README.TXT(374) : Matched "Alpha", replace with "I386"
.\README.TXT
```

Также по умолчанию для всех файлов, в которых произведены замены, создаются резервные копии с расширением bak.

Рассмотрим теперь назначение ключей команды MUNGE.

Ключ -q означает, что нужно выполнять только операцию поиска — замена при этом не производится. Например, задав команду

```
MUNGE string.msk -q .
```

мы на экране получим информацию следующего вида:

```
Reading script file - 1.msk 0.tokens and 2 literal strings
Scanning .\*.*
.\PROBREP.TXT(19) : CD-ROM
.\PROBREP.TXT(94) : CD-ROM CONFIGURATION ******
.\PROBREP.TXT(95) : CD-ROM drive band & model
.\PROBREP.TXT [3 potential changes]
.\README.TXT(13) : Alpha processor architectures:
.\README.TXT(16) : Alpha
.\README.TXT(261) : Alpha) and run SETUP.EXE. Setup will replace the
.\README.TXT(374) : Alpha\Mdac directories and follow
.\README.TXT [4 potential changes]
```

Применение ключей -e и -o, с помощью которых также производится только поиск строк, позволяет менять режим вывода информации: ключ -e позволяет для каждого совпадения выводить целую строку, а с помощью ключа -o можно выводить имя файла лишь для первого совпадения.

При использовании ключа -i выводятся только итоговые изменения в файлах. Например, команда

```
MUNGE string.msk -q -i .
```

предоставит такую информацию:

```
Reading script file - 1.msk 0.tokens and 2 literal strings
Scanning .\*.*
.\PROBREP.TXT [3 potential changes]
.\README.TXT [4 potential changes]
.\README.TXT(1) : 4 changes made to this file
.\PROBREP.TXT(1) : 3 changes made to this file
```

Использование ключа `-m` позволяет сворачивать несколько подряд идущих символов возврата каретки в один.

Применение ключа `-r` указывает на то, что нужно обработать файлы не только в заданном каталоге, но и во всех его подкаталогах (в других командах для этого обычно применяется ключ `/S`).

Использование ключа `-a` в случае необходимости снимает атрибут "только для чтения" с файлов, в которых производится замена строк.

Если вы уверены, что резервные копии файлов, в которых были произведены замены, вам не понадобятся, можно в командной строке указать ключ `-t`. В этом случае `bak`-файлы создаваться не будут.

Использование ключа `-k` позволяет искать строки, указанные в файле с шаблонами, с учетом регистра символов (по умолчанию поиск проводится без учета регистра).

В случае применения ключа `-u` *файл_отмены* автоматически создается файл, в который записываются шаблоны строк, позволяющие сделать обратную замену.

Использование ключа `-v` позволяет выводить развернутую информацию. В этом случае на экран выводятся имена проверяемых файлов.

Работа с переменными среды

В Windows NT различаются два вида переменных среды: переменные *среды операционной системы* (системные переменные) и переменные *среды текущего пользователя*.

Имеется возможность графического просмотра и редактирования переменных обоих типов. Для этого нужно выбрать пункт **Система** (System) в Панели управления (Control Panel) и в диалоговом окне **Свойства системы** (System Properties) открыть вкладку **Переменные среды** (Environment) (рис. 3.4).

Системные переменные среды определяются Windows NT и имеют одни и те же значения, не зависящие от того, какой пользователь вошел на компьютер. Например, переменные `COMSPEC` (полный путь к командному интерпретатору, `COMSPEC=C:\WINNT\system32\cmd.exe`), `OS` (название операционной системы, `OS=Windows_NT`), `WINDIR` (каталог Windows NT, `WINDIR=C:\WINNT`). Добавлять новые системные переменные или изменять значения существующих могут члены группы администраторов.

Переменные среды текущего пользователя могут иметь разные значения для каждого пользователя на конкретном компьютере. В число таких переменных входят переменные, определяемые в приложениях (например, путь к каталогу, в котором сохраняются файлы приложений).

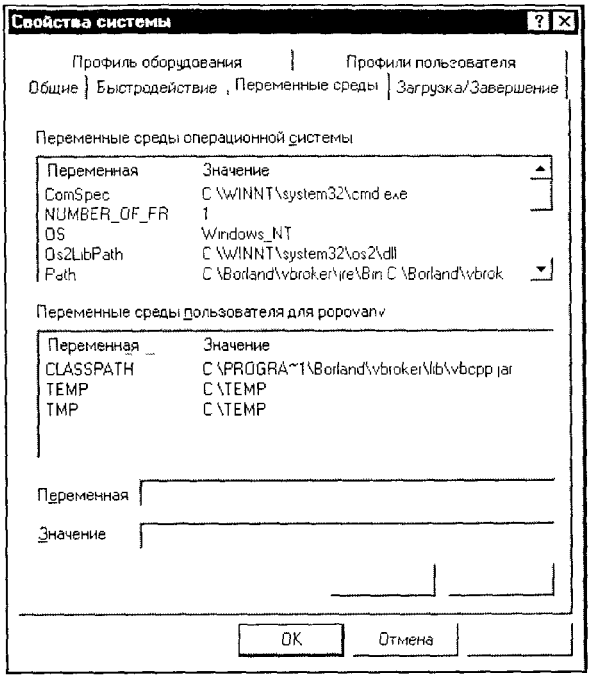


Рис. 3.4. Переменные среды в Windows NT Workstation 4.0

Кроме того, используя команду `SETX` из пакета Windows NT Resource Kit, можно изменить значения переменных среды из командной строки (новые значения переменных будут записаны в системный реестр). Эта команда будет подробно описана ниже.

По умолчанию при загрузке Windows NT находит файл `C:\autoexec.bat`, если он есть, и берет оттуда все установки для переменных среды. Скажем, если в `autoexec.bat` определена переменная `PATH`, то путь, задаваемый этой переменной, будет добавляться к системному пути по умолчанию каждый раз, когда какой-либо пользователь регистрируется в системе.

Когда открывается новое окно командного интерпретатора, переменные среды обоих типов копируются в переменные среды этого командного окна. При этом копирование происходит в следующем порядке:

1. Переменные из файла `autoexec.bat`.
2. Переменные среды операционной системы.
3. Переменные среды пользователя.
4. Переменные из файла `%SystemRoot%\SYSTEM32\autoexec.nt` (этот файл будет описан в *гл. 4*).

Например, если в файле autoexec.bat имеется строка

```
SET TMP=C:\
```

и, кроме этого, задана переменная среды пользователя TMP со значением D:\TEMPDIR, то значением переменной TMP в командном окне будет D:\TEMPDIR (конечно, если только значение переменной TMP не переопределяется еще раз в файле autoexec.nt).

Команда *SET*

Работа с переменными среды текущего командного окна осуществляется, как и в Windows 9*x*, с помощью команды SET. Естественно, изменения, которые вносятся в переменные среды этой команды, актуальны только в текущем командном окне.

Если режим расширенной обработки команд не включен, то синтаксис команды SET остается прежним:

```
SET [переменная=[строка]]
```

В частности, команда SET, запущенная без параметров, выводит значения всех переменных среды текущего командного окна.

Новые, весьма полезные, возможности у команды SET появляются при включении расширенной обработки команд. Остановимся подробнее на этих возможностях.

Если при вызове команды SET указать только имя переменной без знака равенства и значения, то команда выведет значения всех переменных, имя которых начинается с указанной строки. Таким образом, команда

```
SET P
```

отобразит значения всех переменных, имена которых начинаются с P (PATH, например).

Если имя переменной не найдено в текущей среде, то при возврате команда SET установит значение ERRORLEVEL равным 1. Это свойство можно использовать в командных файлах для определения наличия определенной переменной.

Команда SET допускает использование знака равенства (=) в любой позиции значения переменной среды, кроме первого символа.

Самым же главным новшеством является то, что теперь переменные могут рассматриваться как числа и с ними можно производить арифметические вычисления. Для этой цели в команде SET имеется дополнительный ключ /A:

```
SET /A переменная=выражение
```

Использование ключа /A указывает, что стоящая справа от знака равенства строка является числовым выражением, значение которого вычисляется.

Например, если задать команду

```
SET /A M=1+2
```

то значение переменной M будет равно трем (M=3). Обработчик выражений, входящих в команду SET, очень прост и поддерживает следующие операции, перечисленные в порядке убывания приоритета:

- группировка с помощью круглых скобок ();
- арифметические операторы умножения (*), целочисленного деления (/), остатка от деления (%);
- арифметические операторы сложения (+) и вычитания (-);
- двоичный сдвиг влево (<<) и вправо (>>);
- двоичное И (&);
- двоичное исключающее ИЛИ (^);
- двоичное ИЛИ (|);
- операторы присваивания =*, =/, =%, =+, =-, =, &=, ^=, |=, <<= и >>=;
- разделение операторов с помощью запятой (,).

При использовании любых логических или двоичных операторов необходимо заключить строку выражения в кавычки. Любые нечисловые строки в выражении рассматриваются как имена переменных среды, значения которых преобразуются в числовой вид перед использованием. Если переменная с указанным именем не определена в системе, вместо нее подставляется нулевое значение. Например, если переменная среды X не была предварительно задана, то в результате выполнения команды

```
SET /A N=X+5
```

значение переменной N будет равно пяти (N=5).

Таким образом, применение ключа /A позволяет выполнять арифметические операции со значениями переменных среды, причем не нужно вводить знаки % для получения их значений. Например:

```
SET /A M=1
SET /A N=M+1
```

Если команда SET /A вызывается из командной строки, а не из пакетного файла, то она выводит на экран окончательное значение вычисленного выражения. Слева от любого оператора присваивания должно стоять имя переменной среды.

Числовые значения в команде SET могут рассматриваться как десятичные (по умолчанию), шестнадцатеричные (префикс 0x), двоичные (префикс 0b) или восьмеричные (префикс 0) числа. Например, числа 0x12, 0b10010 и 022 обозначают десятичное число 18.

В Windows NT также немного усовершенствована работа с переменными среды как со строками. Например, следующая команда:

```
SET переменная1=%переменная2:строка1=строка2%
```

раскроет значение второй указанной переменной среды (*переменная2*), заменит там все вхождения *строки1* на *строку2* и запишет результат в первую переменную (*переменная1*). Скажем, если значением переменной S была строка D:\Programs\Aditor, то в результате выполнения команды

```
SET S=%S:Programs=Программы%
```

переменная S приобретет значение D:\Программы\Aditor. Подставляемая строка (*строка2*) может быть пустой, что приведет к удалению всех вхождений *строки1* из раскрытого значения переменной. Если *строка1* начинается со знака звездочки, то будет заменен весь текст с начала раскрытого значения до первого вхождения оставшейся части *строки1*. Например, если выполнить следующие команды:

```
SET S=Раз Два Три Раз Два Три
SET M=%S:* Три=Четыре%
```

то значением переменной M будет строка Четыре Раз Два Три.

Если задать команду вида

```
SET переменная1=%переменная2:~m,n%
```

то она раскроет значение второй указанной переменной (*переменная2*), использует из него только n символов, начиная с (m+1)-го (m определяет количество символов, на которое происходит сдвиг) и запишет результат в первую переменную (*переменная1*). Например, в результате выполнения следующих команд:

```
SET S=Раз Два Три Раз Два Три
SET M=%S:~4,3%
```

значением переменной M будет слово Два.

Команда *SETX*

Рассмотрим теперь команду SETX из пакета Windows NT Resource Kit, которая позволяет непосредственно устанавливать значения переменных среды пользователя или среды операционной системы (еще раз напомним, что команда SET позволяет изменять переменные только в среде текущего командного окна). Эта команда является единственным средством для изменения значений переменных среды Windows NT из командной строки.

Команда SETX позволяет устанавливать значения переменных среды из трех источников: командной строки, системного реестра и текстового файла.

В соответствии с этим будем говорить о трех режимах работы команды SETX:

1. Режим командной строки.
2. Режим реестра.
3. Файловый режим.

Самым простым является синтаксис утилиты для режима командной строки:

```
SETX переменная значение [-M]
```

Использование ключа -M позволяет устанавливать значение переменной в окружении операционной системы (по умолчанию значение устанавливается в окружении пользователя).

Синтаксис для режима реестра имеет вид:

```
SETX переменная -K корневой_ключ\ключ\...\переменная [-M]
```

Применение ключа -K указывает на то, что значение устанавливаемой переменной будет взято из ключа реестра, путь к которому определяется параметром *корневой_ключ\ключ\...\значение*.

Использование ключа -M устанавливает значение переменной в окружении операционной системы (по умолчанию значение устанавливается в окружении пользователя).

Синтаксис для файлового режима имеет вид:

```
SETX переменная -F имя_файла {-A x,y|-R x,y "строка"} [-D d] [-X] [-M]
```

Применение ключа -F *имя_файла* задает имя файла, из которого будет считываться информация.

Использование ключа -A x,y задает абсолютные координаты и смещение для поиска последовательности символов в файле, указанном с помощью ключа -F: параметр x означает номер строки в файле, параметр y — номер слова в этой строке.

Замечание

Нумерация строк и слов начинается с нуля, а не с единицы.

Например, если имеется текстовый файл 1.txt следующего содержания:

```
Value1 Value2
Value3 Value4
```

то после выполнения команды

```
SETX MyVar -F 1.txt -A 1,0
```

в переменную MyVar запишется значение Value3 (нулевое слово в первой строке).

Указание ключа `-R x,y "строка"` задает в качестве параметров поиска относительные координаты и смещение от заданной строки. Например, после выполнения команды

```
SETX MyVar -F 1.txt -R 0,1 "Value1"
```

в переменную `MyVar` запишется значение `Value2` (смещение на одно слово вправо от строки `Value1`). Применение ключа `-D d` определяет дополнительные разделители (d), например "," или "\".

По умолчанию команда `SETX` воспринимает четыре разделителя: пробел, буляцию, возврат каретки и перевод строки. В качестве дополнительн разделителей можно использовать любой ASCII-символ. Максимальное чи ло разделителей, включая четыре встроенных, равно 15.

Указание ключа `-X` обеспечивает вывод координат в файле, при этом ключи `-A`, `-R` и `-D` игнорируются. Например, после выполнения команды

```
SETX MyVar -F 1.txt -X
```

на экран будут выведены следующие строки:

```
(0,0 Value1) (0,1 Value2)
(1,0 Value3) (1,1 Value4)
```

Использование ключа `-M` устанавливает значение переменной в окружении операционной системы (по умолчанию значение устанавливается в окружении пользователя).

Отметим, что команда `SETX` имеет некоторые ограничения.

- Эта команда записывает значения устанавливаемых переменных в системный реестр, поэтому переменные, установленные с помощью `SETX` будут доступны только в следующем командном окне, а не в текущем (это свойство операционной системы Windows NT).
- В режиме реестра можно считывать данные только из корневых ключей `HKEY_CURRENT_USER` и `HKEY_LOCAL_MACHINE`.
- Если команда `SETX` считывает из реестра данные типа `REG_MULTI_SZ` (множество различных значений, в качестве разделителя которых используется символ ASCII 0), то используется только первый элемент из множества.
- Ключи реестра типа `REG_DWORD` распаковываются и используются в шестнадцатеричном формате.
- Переменные, которые были добавлены в окружение операционной системы или в окружение пользователя, не могут быть удалены с помощью команды `SETX` (в случае переменных текущего командного окна это можно было сделать, указав команду `SET` с именем переменной, но с пустым значением).

- В файловом режиме утилита SETX может разбирать только текстовый файл, строки которого разделяются символами CR-LF (возврат каретки и перевод строки).

Запуск программ и документов

В Windows NT приложения и документы зарегистрированных типов запускаются, как и в Windows 9*x*, с помощью команды START. Однако в Windows NT эта команда имеет намного больше ключей и обладает более богатыми возможностями по управлению запускаемым процессом. Кроме того, в Windows NT имеются команды, с помощью которых можно зарегистрировать тип файлов с заданным расширением и сопоставить файлам определенного типа команду для их открытия (в Windows 9*x* эти операции можно было проделать только с помощью Проводника Windows).

Отметим, что при включении расширенной обработки команд для вызова неисполняемых файлов через механизм сопоставления типов файлов необязательно использовать команду START, достаточно просто ввести имя файла в командной строке и нажать клавишу <Enter>. Например, команда ACCOUNT.TXT сразу запускает приложение, сопоставленное расширению txt, и загружает в него файл account.txt.

Синтаксис команды START в Windows NT имеет следующий вид:

```
START ["заголовок"] [/Dпуть] [/I] [/MIN] [/MAX] [/SEPARATE|/SHARED] [/LOW|
/NORMAL|/HIGH|/REALTIME] [/WAIT] [/B] [команда/программа] [параметры]
```

Если параметр *команда/программа* определяет внутреннюю команду интерпретатора cmd.exe или пакетный файл, то для их выполнения в новом (если не указан ключ /B) окне автоматически запускается интерпретатор команд cmd.exe с ключом /K. Таким образом, в этом случае новое окно не будет закрыто после завершения команды. Если же запускается не внутренняя команда cmd.exe и не пакетный файл, то эта программа запускается в графическом или текстовом окне.

В случае, когда первым элементом командной строки является слово CMD без расширения и пути к файлу, обработчик команд перед выполнением строки заменяет слово CMD на значение переменной COMSPEC (полный путь к командному интерпретатору), что позволяет избежать неожиданного запуска случайных версий файла cmd.exe.

Если имя запускаемой программы задано без расширения, то командный интерпретатор использует значение переменной среды PATHEXT (в Windows 9*x* такой переменной нет), чтобы определить расширения имен исполняемых файлов и порядок поиска нужного файла. По умолчанию для переменной PATHEXT задается следующее значение:

```
PATHEXT=.COM;.EXE;.BAT;.CMD
```

(cmd — это расширение для командных файлов в Windows NT). Здесь синтаксис подобен синтаксису для переменной PATH, т. е. отдельные элементы разделяются точкой с запятой. Если ни одного файла с заданными по умолчанию расширениями не найдено, интерпретатор команд проверяет, задает ли указанное имя существующий каталог. Если это так, то команда START запускает Проводник Windows и открывает в нем указанный каталог.

Параметр *заголовок* в команде START определяет заголовок создаваемого окна. Например:

```
START "Копирование данных" copier.bat
```

Если команда START открывает новое командное окно, то в нем можно сразу указать рабочий каталог. Это делается с помощью параметра *путь*.

Применение ключа /I означает, что новой операционной средой станет исходная среда, переданная командным интерпретатором cmd.exe, а не текущая среда командного окна.

Ключи /MIN и /MAX имеют те же значения, что и в Windows 9*x*: если указан ключ /MIN, то запуск команды/программы происходит в свернутом окне, если /MAX — то в развернутом (максимизированном) окне.

Ключи /SEPARATE и /SHARED используются для указания режима запуска 16-разрядных приложений Windows. Если указан ключ /SEPARATE, то запуск такой программы происходит в отдельной области памяти, если /SHARED — то в общей области памяти.

Следующие четыре ключа отвечают за приоритет запускаемой задачи. Применение ключа /LOW означает, что приложение запускается с приоритетом IDLE, ключа /NORMAL — с приоритетом NORMAL, ключа /HIGH — с приоритетом HIGH, ключа /REALTIME — с приоритетом REALTIME.

Ключ /WAIT используется для запуска приложения с ожиданием его завершения.

Если указан ключ /B, то запуск приложения происходит без создания нового окна (конечно, если это возможно). Таким образом, если с этим ключом запускается внутренняя команда cmd.exe или пакетный файл, то новая копия командного интерпретатора будет запущена в текущем командном окне.

Команды *ASSOC* и *FTYPE*

Рассмотрим теперь, каким образом можно из командной строки сопоставить файлам с определенным расширением программу, которая будет их открывать. Для этого необходимо использовать две команды: ASSOC и FTYPE.

С помощью ASSOC можно устанавливать или изменять связи между расширениями и типами файлов. Синтаксис этой команды похож на синтаксис команды SET:

```
ASSOC [.рсш[=[тип_файла]]]
```

Параметр *.расш* здесь задает расширение для связи с типом файлов, а *тип_файла* указывает тип файла для связи с данным расширением. Например:

```
ASSOC .txt=txtfile
```

Команда ASSOC, запущенная без параметров, выведет информацию обо всех текущих связях между расширениями и типами файлов. Для того чтобы вывести информацию только о типе файлов с одним заданным расширением, нужно использовать команду ASSOC *.расш*, например

```
ASSOC .txt
```

Команда ASSOC, заданная без параметра *тип_файла*, удалит связь между данным расширением и типом файлов, например

```
ASSOC .txt=
```

С помощью команды FTYPE можно сопоставить определенному типу файлов программу, которая будет их открывать. Синтаксис этой команды имеет вид:

```
FTYPE [тип_файлов[=[командная_строка_открытия]]]
```

Здесь параметр *командная_строка_открытия* задает команду открытия, использующуюся при запуске файлов указанного типа.

Команда FTYPE без параметров выводит список всех типов файлов, для которых определены командные строки открытия. Если указан только тип файла, FTYPE выводит командную строку открытия для этого типа файлов. Например, если задать команду

```
FTYPE txtfile
```

то на экран выведется строка следующего вида:

```
txtfile=C:\aditor\aditor.exe "%1"
```

Если после знака равенства не указана строка открытия, FTYPE удалит текущее сопоставление для указанного типа файлов.

В параметре *командная_строка_открытия* можно использовать замещаемые параметры командной строки %0 — %9. Когда будет запускаться программа, сопоставленная заданному типу файлов, переменные %0 и %1 заменяются на имя файла, запускаемого с помощью сопоставления. Вместо переменной %* подставляются все оставшиеся параметры командной строки, а переменные %2, %3 и т. д. заменяются, соответственно, на первый, второй и другие параметры. Вместо переменной %~n подставляются все оставшиеся параметры, начиная с n, где n является числом от 2 до 9. Например:

```
ASSOC .pl=PerlScript
FTYPE PerlScript=perl.exe %1 %*
```

Эти команды позволят вызывать из командной строки интерпретатор сценариев Perl следующим образом:

```
script.pl 1 2 3
```

Более того, если записать расширение pl в переменную среды PATHEXT, то можно не вводить расширение pl в именах файлов. Для этого команда SET используется следующим образом:

```
SET PATHEXT=.pl;%PATHEXT%
```

Теперь обработчик сценариев Perl вызывается еще проще:

```
script 1 2 3
```

Сетевые команды

В Windows NT имеется намного больше команд для работы с локальной сетью, чем в Windows 9*x* (полный список этих команд приведен в *приложении 1*). В табл. 3.6 кратко описаны основные сетевые команды Windows NT, начинающиеся со слова NET.

Таблица 3.6. *Основные сетевые команды Windows NT (NET...)*

Команда	Описание
NET ACCOUNTS	Обновление базы учетных данных пользователей, изменение паролей и параметров подключения для всех пользователей
NET COMPUTER	Добавление или удаление имени компьютера в базе данн домена. Эта команда доступна только на компьютерах с оп рационной системой Windows NT Server
NET CONFIG	Вывод информации о запущенных службах или изменени настроек служб
NET CONTINUE	Активация приостановленной службы
NET FILE	Вывод имен открытых общих файлов на сервере и количес ва блокировок для каждого файла, если они установлены
NET GROUP	Вывод, изменение и удаление глобальных групп в домен сервера Windows NT. Команда доступна только в домен сервера Windows NT
NET HELP	Вывод списка доступных сетевых команд и тем, по которым может быть получена справка, или информации по конкрет ной команде или теме
NET HELPMSG	Получение справки о сообщениях об ошибках системы Windows NT

Таблица 3.6 *(окончание)*

Команда	Описание
NET LOCALGROUP	Добавление, удаление и вывод информации о локальных группах
NET NAME	Добавление или удаление синонима или вывод списка имен, под которыми компьютер принимает сообщения
NET PAUSE	Приостановка работающей службы
NET PRINT	Управление заданиями и очередями принтеров и вывод информации об этих очередях
NET SEND	Посылка сообщения другому пользователю, компьютеру или синониму в сети
NET SESSION	Отключение пользователей, подсоединенных к компьютеру или вывод списка таких пользователей
NET SHARE	Создание и удаление совместно используемых ресурсов или вывод информации о ресурсах
NET START	Запуск службы или вывод списка запущенных служб
NET STATISTICS	Вывод статистической информации локальной службы сервера или рабочей станции
NET STOP	Остановка сетевой службы Windows NT
NET TIME	Синхронизация часов компьютера с часами другого компьютера или домена
NET USE	Подключение к общим сетевым ресурсам или вывод информации о подключениях компьютера. Команда также управляет постоянными сетевыми соединениями
NET USER	Добавление, редактирование или просмотр учетной информации пользователя
NET VIEW	Просмотр списка доменов, компьютеров или общих ресурсов на данном компьютере

Кроме этого, в составе Windows NT имеются специальные утилиты для работы с протоколом TCP/IP (например, ARP, PING, ROUTE); эти утилиты здесь описываться не будут.

В данном разделе мы рассмотрим, какие изменения были внесены в две команды, которые были описаны в *гл. 1*, — NET VIEW и NET USE. После этого разберем еще две новые команды: NET SHARE и NET USER. Также будет описана команда PERMCOPY из пакета Windows NT Resource Kit.

Команды *NET VIEW* и *NET USE*

В синтаксисе команды NET VIEW, предназначенной для просмотра списк доменов, компьютеров или общих ресурсов на данном компьютере, пр изошли небольшие изменения, и теперь он имеет вид:

```
NET VIEW [\\компьютер|/DOMAIN[:домен]]
NET VIEW /NETWORK:NW [\\компьютер]
```

Как мы видим, новыми здесь являются ключи /DOMAIN и /NETWORK.

Использование ключа /DOMAIN:*домен* задает домен, для которого выводится список компьютеров. Если параметр *домен* не задан, то выводится информация обо всех доменах в сети (в Windows 9*x* такой возможности не было).

С помощью указания ключа /NETWORK:NW выводятся все доступные ресурсы серверов сети NetWare. Если задано имя компьютера, то выводится список ресурсов данного компьютера в сети NetWare.

Команда NET USE в Windows NT также обладает некоторыми новыми возможностями, в частности с ее помощью можно создавать постоянные сетевые соединения (не исчезающие после перезагрузки компьютера).

Поддерживаются три варианта синтаксиса этой команды:

```
NET USE  [устройство|*] [\\компьютер\ресурс[\папка]][пароль|*]]
[/USER:[домен\]пользователь][[/DELETE]|[/PERSISTENT:{YES | NO}]]
NET USE устройство [/HOME[пароль|*]] [/DELETE:{YES|NO}]
NET USE [/PERSISTENT:{YES|NO}]
```

Применение ключа /USER позволяет задать иное имя пользователя для подключения к общему ресурсу. Параметр *домен* задает имя другого домена, а *пользователь* — имя пользователя для подключения. Например, команда

```
NET USE H: \\Server1\Share /USER:Sklad\ivanov
```

подключает к общему ресурсу \\Server1\Share пользователя ivanov из дом на Sklad. Если имя домена в ключе /USER не задано, то используется тек щий домен.

Использование ключа /PERSISTENT позволяет управлять постоянными се выми подключениями (подключения без устройств не являются постоянн ми). Параметр YES в этом ключе означает, что все существующие соедин ния будут сохранены и восстановятся при следующем подключении. Па метр NO указывает на то, что заданное подключение не будет сохранено. Д удаления постоянных подключений используется ключ /DELETE.

Для восстановления текущих подключений при следующих входах в се независимо от будущих изменений, служит команда:

```
NET USE /PERSISTENT:YES
```

Команда *NET SHARE*

С помощью команды NET SHARE можно создавать и удалять совместно используемые ресурсы, а также выводить информацию о таких ресурсах.

Команда NET SHARE, запущенная без параметров, выводит информацию обо всех совместно используемых ресурсах на локальном компьютере. Когда выводится список общих ресурсов компьютера, то для каждого ресурса будет выведено сетевое имя ресурса, имя устройства или путь, соответствующие данному ресурсу, и комментарий (если он имеется). Вывод команды NET SHARE имеет следующий вид:

```
Общее имя    Ресурс                  Заметки
-------------------------------------------------------------------------
IPC$                                 Удаленный IPC
C$           C:\                     Стандартный общий ресурс
D$           D:\                     Стандартный общий ресурс
ADMIN$       C:\WINNT                Удаленный Admin
TeX          C:\TeX                  MikTeX
```

Команда

```
NET SHARE ресурс,
```

где параметр *ресурс* соответствует сетевому имени общего ресурса, выведет информацию об отдельном ресурсе. Например, команда

```
NET SHARE MikTeX
```

выводит информацию в следующем виде:

```
Общее имя                    MikTeX
Путь                         C:\TeX
Заметки                      TeX под Windows
Макс. число пользователей    10
Пользователи
```

Создать новый совместно используемый ресурс можно с помощью следующей команды:

```
NET SHARE ресурс=диск:путь [/USERS:количество|/UNLIMITED] [/REMARK:"текст"]
```

Параметр *диск:путь* задает абсолютный путь к каталогу, который нужно предоставить в общее пользование. Для совместного использования каталога, в имени которого содержатся пробелы, диск и путь этого каталога должны быть заключены в кавычки (" "), например:

```
NET SHARE MF="C:\Multimedia Files"
```

Когда общий ресурс создается на сервере, его конфигурация сохраняется. После остановки службы сервера все общие ресурсы отключаются, но после следующего запуска службы сервера они будут восстановлены.

Ключ /USERS:*количество* задает максимальное количество пользователей, ко торым разрешен одновременный доступ к общему ресурсу. Для того чтоб отменить ограничение на число пользователей, которым разрешен одновре менный доступ к общему ресурсу, необходимо использовать ключ /UNLIMITED.

Комментарий к ресурсу добавляется при использовании ключа /REMARK:"*текст*".

Изменить свойства уже созданного общего ресурса можно с помощью сле дующего варианта команды NET SHARE:

```
NET SHARE ресурс [/USERS:количество|/UNLIMITED][/REMARK:"текст"]
```

Существующий общий ресурс можно удалить (сделать локальным). Для этого нужно знать либо его сетевое имя, либо абсолютный путь к предос тавленному в совместное использования каталогу, и задать команду NET SHARE следующим образом:

```
NET SHARE {ресурс|диск:путь} /DELETE
```

Рассмотрим примеры использования команды NET SHARE.

1. Для вывода информации обо всех общих ресурсах на компьютере в файл share.txt служит следующая команда:

   ```
   NET SHARE > share.txt
   ```

2. Для предоставления в совместное использование каталога C:\LETTERS под сетевым именем SECRETARY и задания комментария к этому ресурсу слу жит следующая команда:

   ```
   NET SHARE SECRETARY=C:\LETTERS /REMARK:"Переписка."
   ```

   Для прекращения совместного использования каталога C:\LETTERS служит команда

   ```
   NET SHARE SECRETARY /DELETE
   ```

   или

   ```
   NET SHARE C:\LETTERS /DELETE
   ```

3. Для задания совместного использования каталога C:\Мои документы под именем DOCUM можно использовать следующую команду (обратите внима ние на кавычки):

   ```
   NET SHARE DOCUM=" C:\Мои документы"
   ```

Команда *PERMCOPY*

С помощью команды PERMCOPY из пакета Windows NT Resource KIT можно быстро скопировать права доступа с одного общего ресурса на другой. Син таксис этой команды имеет вид:

```
PERMCOPY \\источник ресурс_источник \\результат ресурс_результат
```

Например, чтобы скопировать права доступа с общего каталога `Programs` на сервере `\\Server1` на общий каталог `Documents` на сервере `\\Server2`, нужно использовать следующую команду:

```
PERMCOPY \\Server1 Programs \\Server2 Documents
```

Отметим, что у команды `PERMCOPY` есть одно ограничение — с ее помощью нельзя скопировать на общий ресурс права доступа с ресурса, предназначенного для административных нужд (например, с ресурса `C$`).

Одной из самых распространенных задач администрирования в Windows NT (как и в любой другой многопользовательской операционной системе) является работа с учетными записями пользователей. Напомним, что стандартной графической программой, предназначенной для этой цели, является **Диспетчер пользователей** (User Manager for Domain), который находится в меню **Пуск/Программы** (Start/Programs). Окно этой программы приведено на рис. 3.5.

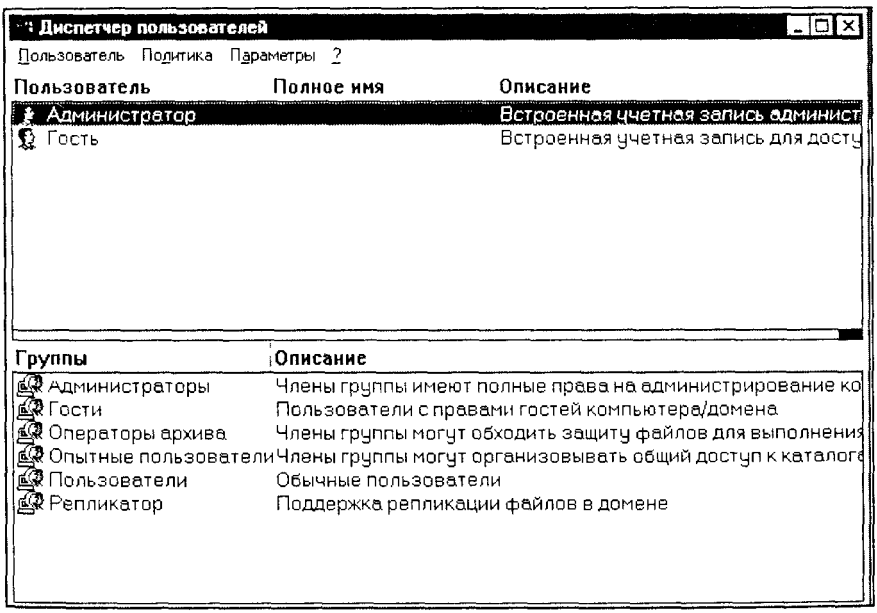


Рис. 3.5. Диспетчер пользователей в Windows NT Workstation 4.0

Команда *NET USER*

Добавлять, редактировать или просматривать учетную информацию заданного пользователя из командной строки можно с помощью команды `NET USER` (или `NET USERS`). Использование этой команды в пакетных файлах может намного облегчить процедуру ввода учетных записей пользователей.

Команда NET USER может задаваться в трех вариантах:

```
NET USER [пользователь [пароль|*] [параметры]] [/DOMAIN]
NET USER пользователь {пароль|*} /ADD [параметры] [/DOMAIN]
NET USER пользователь [/DELETE] [/DOMAIN]
```

Запущенная без параметров утилита NET USER выводит список всех пользователей, зарегистрированных на данном компьютере. Используя символы перенаправления > и >>, этот список можно направить в текстовый файл.

Параметр *пользователь* обозначает логическое имя пользователя, учетная запись которого добавляется, редактируется или просматривается. Имя может иметь до 20 символов в длину. При запуске команды NET USER на сервере Windows NT Server все изменения автоматически вносятся в базу данных основного контроллера домена и дублируются на резервных контроллерах.

Параметр *пароль* позволяет производить присвоение или изменение пароля пользователя. Минимальная длина пароля должна соответствовать ограничению, установленному ключом /MINPWLEN команды NET ACCOUNTS. Пароль может иметь длину до 14 символов. Символ *, указанный вместо пароля, обеспечивает вывод приглашения для ввода пароля. При вводе пароля с клавиатуры символы на экран не выводятся.

Применение ключа /DOMAIN означает, что операции осуществляются на основном контроллере основного домена, которому принадлежит компьютер. Этот параметр имеет смысл только на рабочих станциях Windows NT, подключенных к домену сервера Windows NT. Сервер Windows NT, по умолчанию, осуществляет операции на основном контроллере домена.

Использование ключа /ADD позволяет добавить пользователя в базу данных учетной информации, а ключа /DELETE — удалить информацию о пользователе из этой базы данных.

Параметры команды NET USER могут состоять из одного или нескольких ключей, описанных ниже.

Использование ключа /ACTIVE:{NO|YES} позволяет блокировать (NO) или активизировать (YES) учетное имя пользователя. Если пользователь блокирован, то он не сможет подключиться к ресурсам компьютера. По умолчанию используется значение YES (активен).

Ключ /COMMENT:"*текст*" применяется для задания комментария к пользовательской записи. Длина этого комментария может составлять до 48 символов. Текст заключается в кавычки.

Ключ /COUNTRYCODE:*nnn* используется для указания кода страны, определяющего файлы, которые применяются при выводе справочной информации и сообщений об ошибках. Нулевое значение ключа обозначает код страны, используемый по умолчанию.

Задать период действия учетной информации о пользователе можно с помощью ключа /EXPIRES:{*дата*|NEVER}. Если задан параметр *дата*, то учетная

информация о пользователе будет действовать ограниченный период времени, иначе — без ограничений. Дата может задаваться в формате мм/дд/гг, дд/мм/гг или мм,дд,гг, в зависимости от кода страны. Следует иметь в виду, что дата задает день, когда срок действия учетной записи пользователя истечет. Месяц может задаваться числом, полным именем или трехбуквенным сокращением. Год может вводиться двумя или четырьмя цифрами. В качестве разделителей следует использовать запятую или косую черту, но не пробелы.

Применение ключа /FULLNAME:"*имя*" задает действительное имя пользователя, а не его логическое имя. При этом заданное имя заключается в кавычки.

Основной (домашний) каталог пользователя можно задать, указав ключ /HOMEDIR:*путь*. Отметим, что параметр *путь* должен задавать существующий каталог.

Используя ключ /HOMEDIRREQ:{YES|NO} можно установить, обязательно (YES) или нет (NO) задание основного каталога пользователя.

Следующие два ключа связаны с паролями пользователей. Ключ /PASSWORDCHG:{YES|NO} устанавливает, допускается (YES) или нет (NO) изменение пароля самим пользователем. По умолчанию используется YES (допускается).

Ключ /PASSWORDREQ:{YES|NO} задает, является обязательным (YES) или нет (NO) использование пароля. По умолчанию пароль обязателен (YES).

При использовании ключа /PROFILEPATH:[*путь*] устанавливается путь для файла профиля подключения пользователя.

Ключ /SCRIPTPATH:*путь* задает путь файла сценария входа (такие сценарии будут рассмотрены в *разд. "Сценарии входа для пользователей" гл. 4*). Параметр *путь* не может быть абсолютным путем, он всегда указывается относительно каталога %systemroot%\SYSTEM32\REPL\IMPORT\SCRIPTS.

Для редактируемого или добавляемого пользователя можно задать интервалы времени, в течение которых ему разрешено подключение к компьютеру. Это делается с помощью ключа /TIMES:{*время*|ALL}. Значение времени может задаваться в формате день[-день], [день[-день]], час[-час], [час[-час]], с интервалом в 1 час. Дни могут быть полными или сокращенными (Пн, Вт, Ср, Чт, Пт, Сб, Вс или M, T, W, Th, F, S, Su, в зависимости от того, какая версия Windows NT установлена: русифицированная или оригинальная). Часы могут быть в 12-ти или 24-часовом формате. Для 12-часового формата используются обозначения AM, PM или A.M., P.M. Параметр ALL отменяет ограничения на время подключения. Пустое значение не позволяет пользователю подключаться никогда. Дни и часы разделяются запятыми, отдельные интервалы — точкой с запятой (например, Пн,4AM-5PM;Вт,1PM-3PM). Пробелы в этом ключе не допускаются.

В учетной информации пользователя может иметься комментарий, которы добавляется или изменяется при использовании ключа /USERCOMMENT:"*текс* Текст комментария нужно вводить в кавычках.

Для заданного пользователя можно указать до восьми имен рабочих ста ций, с которых он может входить в сеть. Для этого применяется клю /WORKSTATIONS:{*компьютер*[,...]|*}. Элементы списка разделяются запятыми. Если в ключе /WORSTATIONS список компьютеров не задан, или задан звездочкой *, то пользователю разрешается входить в сеть с любого компьютера.

Рассмотрим примеры использования команды NET USER.

1. Вывести информацию о пользователе popovav можно следующей коман дой:

```
NET USER popovav
```

2. Для добавления учетной записи пользователя Ивана Иванова с правом на подключение с 8 до 17 часов с понедельника по пятницу с обязательным паролем и полным именем пользователя можно использовать следующую команду:

```
NET USER ivanov ivanov /ADD /PASSWORDREQ:YES /TIMES:Пн-Пт,8:00-17:00
/FULLNAME:"Иван Иванов"
```

   Логическое имя пользователя (ivanov) вводится во второй раз в качестве пароля.

3. Для задания времени подключения с 4 до 17 часов в понедельник, с 13 до 15 часов во вторник и с 8 до 17 часов со среды по пятницу для пользователя ivanov используется следующая команда:

```
NET USER ivanov /TIME:Пн,4:00-17:00;Вт,13:00-15:00;Ср-Пт,8:00-17:00
```

4. Для установки параметра /HOMEDIRREQ равного YES для пользователя ivanov и назначения каталога \\SERVER\USERS\IVANOV в качестве его домашнего каталога следует использовать команду:

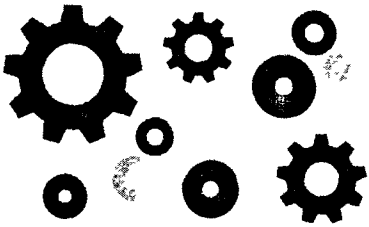
```
NET USER ivanov /HOMEDIRREQ:YES /HOMEDIR:\\SERVER\USERS\HENRYJ
```

Заключение

В Windows NT утилиты командной строки предоставляют намного больше возможностей для управления работой операционной системы, чем в Windows 9*x*, что делает их незаменимым инструментом системного администратора. В данной главе мы рассмотрели лишь некоторые из новых стандартных команд Windows NT и несколько команд из пакета Windows NT Resource Kit, т. к. более-менее полное описание всех этих команд заняло бы не одну сотню страниц.

В гл. 4 будут рассмотрены еще несколько команд, с помощью которых можно создавать более гибкие, чем в Windows 9*x*, и удобные пакетные файлы.

ГЛАВА 4



Командные файлы в Windows NT

В данной главе мы рассмотрим изменения, произошедшие в командах Windows NT, используемых в пакетных файлах, и новые возможности, которые обеспечивают эти команды. Будут описаны следующие стандартные команды Windows NT: `SETLOCAL`, `ENDLOCAL`, `GOTO`, `CALL`, `IF`, `FOR`, `PUSHD`, `POPD`, а также несколько полезных утилит из пакета Windows NT Resource Kit: `FORFILES`, `CHOICE`, `LOGTIME`, `NOW`, `TIMETHIS`, `SLEEP`, `TIMEOUT`, `FREEDISK`, `IFMEMBER`.

Скажем сначала несколько слов об общих особенностях выполнения командных файлов в Windows NT.

Командные файлы в Windows NT, как и в Windows 9*x*, являются обычными текстовыми файлами, однако для них в операционной системе зарезервированы не одно, а два расширения: `bat` и `cmd`. Как уже отмечалось в начале третьей главы, для поддержки приложений MS-DOS (в том числе и пакетных файлов) в Windows NT используется виртуальная машина MS-DOS (NTVDM). При этом для каждого приложения MS-DOS или пакетного файла создается своя NTVDM, свойства которой могут быть настроены путем создания или изменения соответствующего `pif`-файла.

По умолчанию параметры выполнения командного файла (или любой другой DOS-задачи без `pif`-файла) берутся из файла `_default.pif`, находящегося в каталоге `%SystemRoot%` (рис. 4.1).

Замечание

По умолчанию окно, в котором выполняется пакетный файл, закрывается по окончании работы этого файла, что часто бывает неудобно. Для того, чтобы *избежать этого, в окне установки параметров* `_default.pif` нужно снять флажок **Закрывать окно по завершении работы** (Close on exit).

Отметим также, что в Windows NT для каждого приложения или командного файла можно определить собственные конфигурационные загрузочные фай-

лы — аналоги файлов `autoexec.bat` и `config.sys` в MS-DOS и Windows 9x. По умолчанию такими файлами являются `autoexec.nt` и `config.nt`, расположенные в каталоге `%SystemRoot%\SYSTEM32`. Для задания других файлов `autoexec` и `config` предназначена кнопка **Windows NT** на вкладке **Программа** (Program) окна свойств приложения (рис. 4.1). Если нажать эту кнопку, то появится показанное на рис. 4.2 диалоговое окно, в котором можно ввести пути к требуемым конфигурационным файлам.

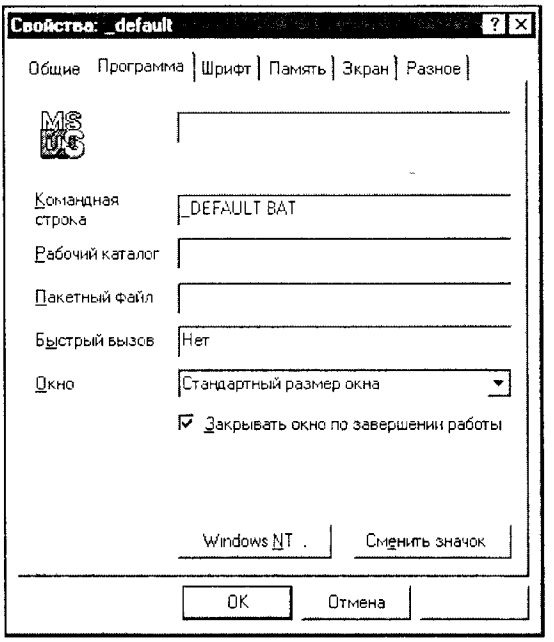


Рис. 4.1. Параметры выполнения командных файлов в Windows NT Workstation 4.0, установленные по умолчанию

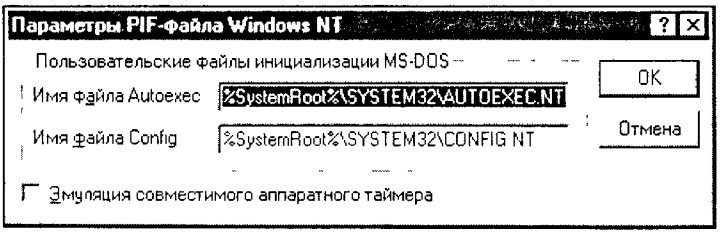


Рис. 4.2. Пути к конфигурационным загрузочным файлам NTVDM

Переменные и параметры командной строки

Поговорим сначала о переменных среды. Как уже отмечалось в третьей главе, в Windows NT при включенной расширенной обработке команд имеется возможность рассматривать переменные среды как числа и производить с ними арифметические вычисления. Для этого используется команда SET с ключом /A *(см. разд. "Работа с переменными среды" гл. 3)*. Приведем пример пакетного файла add.bat, складывающего два числа, заданных в качестве параметров командной строки, и выводящего полученную сумму на экран (листинг 4.1).

Листинг 4.1. Выполнение арифметических действий в командном файле (add.bat)

```
@ECHO OFF
REM В переменной M будет храниться сумма
SET /A M=%1+%2
ECHO Сумма %1 и %2 равна %M%
REM Удалим переменную M
SET M=
```

Напомним, что все изменения, производимые с помощью команды SET над переменными среды в командном файле, сохраняются и после завершения работы этого файла, но действуют только внутри текущего командного окна. В Windows NT имеется возможность локализовать изменения переменных среды внутри пакетного файла, т. е. автоматически восстанавливать значения всех переменных в том виде, в каком они были до начала запуска данного файла. Для этого используются две команды: SETLOCAL и ENDLOCAL. Команда SETLOCAL определяет начало области локальных установок переменных среды. Другими словами, изменения среды, внесенные после выполнения SETLOCAL, будут являться локальными относительно текущего пакетного файла. Каждая команда SETLOCAL должна иметь соответствующую команду ENDLOCAL для восстановления прежних значений переменных среды. Изменения среды, внесенные после выполнения команды ENDLOCAL, уже не являются локальными относительно текущего пакетного файла; их прежние значения не будут восстановлены по завершении выполнения этого файла.

Рассмотрим пример командного файла super.bat, который запускает приложение SuperApp, записывает вывод в файл C:\superapp.out и загружает этот файл в программу Notepad (листинг 4.2).

Листинг 4.2. Файл super.bat

```
@ECHO OFF
REM Запоминаем значения переменных среды
SETLOCAL
REM Добавляем к переменной PATH путь
REM к каталогу G:\programs\superapp
PATH=G:\programs\superapp,%PATH%
REM Запускаем приложение
CALL superapp>c.\superapp.out
REM Восстанавливаем значения переменных среды
ENDLOCAL
REM Загружаем полученный файл в Блокнот
START notepad C:\superapp.out
```

При включении расширенной обработки команд у SETLOCAL можно указывать необязательный аргумент, который может иметь значения ENABLEEXTENSIONS или DISABLEEXTENSIONS Это позволяет временно включить или отключить расширенную обработку команд до выполнения команды ENDLOCAL, независимо от исходного состояния режима обработки команд до вызова команды SETLOCAL

Если команда SETLOCAL вызывается с аргументом, то она устанавливает код ошибки ERRORLEVEL Если указан один из двух допустимых аргументов, то код ошибки будет равен нулю, иначе возвращается значение 1 Это свойство можно использовать в пакетных файлах, чтобы определить доступность расширенной обработки команд, например

```
XCOPY Б: > NUL
SETLOCAL ENABLEEXTENSIONS
IF ERRORLEVEL 1 ECHO Не удается включить расширенную обработку
```

В данном примере команда XCOPY с недопустимым аргументом Б: необходима для установки ненулевого значения ERRORLEVEL, т к в прежних версиях интерпретатора cmd.exe команда SETLOCAL не устанавливает код ошибки

Перейдем теперь к рассмотрению заменяемых параметров командной строки (%0, %1 и т д) В пакетных файлах Windows NT работа с этими параметрами становится более удобной

Во-первых, с помощью символов %* в пакетном файле можно обозначить все аргументы (%1 %2 %3 %4 %5) Таким образом, запустив следующий пакетный файл·

```
@ECHO OFF
ECHO Файл запущен с параметрами: %*
```

с параметрами A B C, мы на экране получим следующее сообщение:

```
Файл запущен с параметрами: A B C
```

Во-вторых, появляются некоторые возможности синтаксического анализа заменяемых параметров. Для параметра с номером n (%n) допустимы синтаксические конструкции (операторы), представленные в табл. 4.1.

Таблица 4.1. *Операторы для заменяемых параметров*

Операторы	Описание
%~Fn	Переменная %n расширяется до полного имени файла
%~Dn	Из переменной %n выделяется только имя диска
%~Pn	Из переменной %n выделяется только путь к файлу
%~Nn	Из переменной %n выделяется только имя файла
%~Xn	Из переменной %n выделяется расширение имени файла
%~Sn	Значение операторов N и X для переменной %n изменяется так, что они работают с кратким именем файла
%~$PATH:n	Проводится поиск по каталогам, заданным в переменной среды PATH, и переменная %n заменяется на полное имя первого найденного файла. Если переменная PATH не определена или в результате поиска не найден ни один файл, эта конструкция заменяется на пустую строку. Естественно, здесь переменную PATH можно заменить на любое другое допустимое значение

Данные синтаксические конструкции можно объединять друг с другом, например:

%~DPn — из переменной %n выделяется имя диска и путь,

%~NXn — из переменной %n выделяется имя файла и расширение.

Рассмотрим следующий пример. Пусть мы находимся в каталоге C:\TEXT и запускаем пакетный файл с параметром Рассказ.doc (%1=Рассказ.doc). Тогда применение операторов, описанных в табл. 4.1, к параметру %1 даст следующие результаты:

```
%~F1=C:\TEXT\Рассказ.doc
%~D1=C:
%~P1=\TEXT\
%~N1=Рассказ
%~X1=.doc
%DP1=C:\TEXT\
%NX1=Рассказ.doc
```

Небольшое изменение произошло также в команде SHIFT, которая производит сдвиг параметров. При включении расширенной обработки команд SHIFT поддерживает ключ /n, задающий начало сдвига параметров с номера n, где n может быть числом от 0 до 9.

Например, в следующей команде:

```
SHIFT /2
```

параметр %2 заменяется на %3, %3 на %4 и т. д., а параметры %0 и %1 остаются без изменений.

Изменения в командах перехода

В расширенном режиме командного интерпретатора cmd.exe в команде перехода внутри файла GOTO можно задавать в качестве метки перехода строку :EOF, которая передает управление в конец текущего пакетного файла. Это позволяет легко выйти из пакетного файла без определения каких-либо меток в самом его конце, например:

```
@ECHO OFF
REM Если файл был запущен без параметров, выходим из него,
REM иначе печатаем первый параметр
IF -%1==- GOTO :EOF
ECHO %1
```

Также при включении расширенной обработки команд произошли изменения в команде вызова внешнего пакетного или исполняемого файла CALL — теперь в качестве адресата вызова в этой команде можно использовать метки внутри текущего командного файла, т. е. применяется следующий синтаксис:

```
CALL :метка аргументы
```

При вызове такой команды создается новый контекст текущего пакетного файла с заданными аргументами и управление передается на инструкцию, расположенную сразу после метки. Для выхода из такого пакетного файла необходимо два раза достичь его конца. Первый выход возвращает управление на инструкцию, расположенную сразу после строки CALL, а второй выход завершает выполнение пакетного файла. Например, если запустить с параметром Копия-1 командный файл следующего содержания:

```
@ECHO OFF
ECHO %1
CALL :2 Копия-2
:2
ECHO %1
```

то на экран выведутся три строки:

```
Копия-1
Копия-2
Копия-1
```

Таким образом, подобное использование команды CALL очень похоже на обычный вызов подпрограмм (процедур) в алгоритмических языках программирования.

Операторы сравнения

В *разд. "Переходы и операторы условия в командных файлах" гл. 2* были рассмотрены три основных варианта команды IF:

```
IF [NOT] строка1==строка2 команда
IF [NOT] EXIST файл команда
IF [NOT] ERRORLEVEL число команда.
```

При включении расширенной обработки команд можно дополнительно применять еще три варианта этой команды:

```
IF [/I] строка1 оператор_сравнения строка2 команда
IF CMDEXTVERSION число команда
IF DEFINED переменная команда
```

Рассмотрим сначала оператор IF в следующем виде:

```
IF [/I] строка1 оператор_сравнения строка2 команда
```

Синтаксис и значение *операторов_сравнения* представлены в табл. 4.2.

Таблица 4.2. *Операторы сравнения в IF*

Оператор	Значение	Оператор	Значение
EQL	Равно	LEQ	Меньше или равно
NEQ	Не равно	GTR	Больше
LSS	Меньше	GEQ	Больше или равно

Приведем пример использования операторов сравнения:

```
@ECHO OFF
CLS
IF -%1 EQL -Вася ECHO Привет, Вася!
IF -%1 NEQ -Вася ECHO Привет, но Вы не Вася!
```

Ключ /I, если он указан, задает сравнение текстовых строк без учета регистра. Ключ /I можно также использовать и в форме *строка1==строка2* команды IF. Например, условие

```
IF /I DOS==dos ...
```

будет истинным.

Отметим также, что сравнение проводится по общему типу данных, так что если обе сравниваемые строки содержат только цифры, то обе строки преобразуются в числа, после чего выполняется сравнение этих чисел. Например, условие

```
IF 002 LEQ 5 ...
```

будет истинным.

Операторы сравнения чисел, приведенные в табл. 4.2, можно применять и в операторе

```
IF ERRORLEVEL ...
```

Например:

```
IF ERRORLEVEL LEQ 1 GOTO Case1
```

При включенной расширенной обработке команд для более удобной работы с кодами завершения программ можно использовать выражение %ERRORLEVEL%. Строка %ERRORLEVEL% будет развернута в строковое представление текущего значения кода ошибки ERRORLEVEL, за исключением ситуации, когда уже имеется переменная среды с именем ERRORLEVEL; в этом случае, естественно, подставляется значение этой переменной. Например, используя данную строку, можно обработать ввод пользователя, полученный с помощью команды CHOICE (напомним, что CHOICE есть только в Windows NT Resource Kit):

```
CHOICE
GOTO Answer%ERRORLEVEL%
:Answer1
ECHO Введено Y (да)
:Answer2
ECHO Введено N (нет)
```

Для определения внутреннего номера версии текущей реализации расширенной обработки команд применяется оператор IF в следующем виде:

```
IF CMDEXTVERSION число команда
```

Здесь условие CMDEXTVERSION применяется подобно условию ERRORLEVEL, но число сравнивается с вышеупомянутым внутренним номером версии. Первая версия имеет номер 1. Номер версии будет увеличиваться на единицу при каждом добавлении существенных возможностей расширенной обра-

ботки команд. Если расширенная обработка команд отключена, условие CMDEXTVERSION никогда не бывает истинно.

Для определения наличия заданной переменной среды предназначен следующий вариант команды IF:

```
IF DEFINED переменная команда
```

Здесь условие DEFINED применяется подобно условию EXISTS наличия заданного файла, но принимает в качестве аргумента имя переменной среды и возвращает истинное значение, если эта переменная определена. Например:

```
@ECHO OFF
CLS
IF DEFINED MyVar GOTO :VarExists
ECHO Переменная MyVar не определена
GOTO :EOF
:VarExists
ECHO Переменная MyVar определена,
ECHO ее значение равно %MyVar%
```

Организация циклов

Напомним, что в Windows 9*x* поддерживался только один вариант команды FOR ... IN ... DO, который обладал довольно ограниченными возможностями. В Windows NT при включенной расширенной обработке команд доступны еще четыре разновидности FOR, которые обеспечивают следующие функции:

- выполнение заданной команды для всех подходящих имен каталогов;
- выполнение заданной команды для определенного каталога, а также всех его подкаталогов;
- получение последовательности чисел с заданными началом, концом и шагом приращения;
- чтение и обработка строк из текстового файла;
- обработка строк вывода определенной команды.

Кроме того, в пакете Windows NT Resource Kit имеется довольно удобная команда FORFILES, предназначенная для выбора файлов в одном или нескольких каталогах и выполнения над ними каких-либо действий.

Рассмотрим подробно дополнительные варианты команды FOR. Первый из них реализуется, если указать в команде FOR ключ /D:

```
FOR /D %переменная IN (набор) DO команда [параметры]
```

В случае если набор содержит подстановочные знаки, то команда выполняется для всех подходящих имен каталогов, а не имен файлов. Скажем, выполнив следующий командный файл:

```
@ECHO OFF
CLS
FOR /D %%f IN (C:\*.*) DO ECHO %%f
```

мы получим список всех каталогов на диске C:, например:

```
C:\Arc
C:\CYR
C:\MSCAN
C:\NC
C:\Program Files
C:\TEMP
C:\TeX
C:\WINNT
```

С помощью ключа /R можно задать рекурсию в команде FOR:

```
FOR /R [[диск:]путь] %переменная IN (набор) DO команда [параметры]
```

В этом случае заданная команда выполняется для каталога [*диск*:]*путь*, а также для всех подкаталогов этого пути. Если после ключа /R не указано имя каталога, то выполнение команды начинается с текущего каталога. Например, для распечатки всех файлов с расширением txt в текущем каталоге и всех его подкаталогах можно использовать следующий пакетный файл:

```
@ECHO OFF
CLS
FOR /R %%f IN (*.txt) DO PRINT %%f
```

Если вместо набора указана только точка (.), то команда проверяет все подкаталоги текущего каталога. Например, если мы находимся в каталоге C:\TEXT с двумя подкаталогами BOOKS и ARTICLES, то в результате выполнения файла:

```
@ECHO OFF
CLS
FOR /R %%f IN (.) DO ECHO %%f
```

на экран выведутся три строки:

```
C:\TEXT\.
C:\TEXT\BOOKS\.
C:\TEXT\ARTICLES\.
```

Ключ /L позволяет реализовать с помощью команды FOR арифметический цикл, в этом случае синтаксис имеет следующий вид:

```
FOR /L %переменная IN (начало,шаг,конец) DO команда [параметры]
```

Здесь заданная после ключевого слова IN тройка (*начало,шаг,конец*) раскрывается в последовательность чисел с заданными началом, концом и шагом приращения. Так, набор (1,1,5) раскрывается в (1 2 3 4 5), а набор (5,-1,1) заменяется на (5 4 3 2 1). Например, в результате выполнения следующего командного файла:

```
@ECHO OFF
CLS
FOR /L %%f IN (1,1,5) DO ECHO %%f
```

переменная цикла %%f пробежит значения от 1 до 5, и на экране напечатаются пять чисел:

```
1
2
3
4
5
```

Числа, получаемые в результате выполнения цикла FOR /L, можно использовать в арифметических вычислениях. Рассмотрим командный файл my.bat следующего содержания:

```
@ECHO OFF
CLS
FOR /L %%f IN (1,1,5) DO CALL :2 %%f
GOTO :EOF
:2
SET /A M=10*%1
ECHO 10*%1=%M%
```

В третьей строке в цикле происходит вызов нового контекста файла my.bat с текущим значением переменной цикла %%f в качестве параметра командной строки, причем управление передается на метку :2 (*см. описание CALL в разд. "Изменения в командах перехода" ранее в этой главе*). В шестой строке переменная цикла умножается на десять, и результат записывается в переменную M. Таким образом, в результате выполнения этого файла выведется следующая информация:

```
10*1=10
10*2=20
10*3=30
10*4=40
10*5=50
```

Самые мощные возможности (и одновременно самый запутанный синтаксис) имеет команда FOR с ключом /F:

```
FOR /F ["ключи"] %переменная IN (набор) DO команда [параметры]
```

Здесь параметр *набор* содержит имена одного или нескольких файлов, которые по очереди открываются, читаются и обрабатываются. Обработка состоит в чтении файла, разбиении его на отдельные строки текста и выделении из каждой строки заданного числа подстрок. Затем найденная подстрока используется в качестве значения переменной при выполнении основного тела цикла (заданной команды).

По умолчанию ключ /F выделяет из каждой строки файла первое слово, очищенное от окружающих его пробелов. Пустые строки в файле пропускаются. Необязательный параметр "*ключи*" служит для переопределения заданных по умолчанию правил обработки строк. Ключи представляют собой заключенную в кавычки строку, содержащую приведенные в табл. 4.3 ключевые слова:

Таблица 4.3. *Ключи в команде* FOR /F

Ключ	Описание
EOL=C	Определение символа комментариев в начале строки (допускается задание только одного символа)
SKIP=N	Число пропускаемых при обработке строк в начале файла
DELIMS=XXX	Определение набора разделителей для замены заданных по умолчанию пробела и знака табуляции
TOKENS=X,Y,M-N	Определение номеров подстрок, выделяемых из каждой строки файла и передаваемых для выполнения в тело цикла

При использовании ключа TOKENS=X,Y,M-N создаются дополнительные переменные. Формат M-N представляет собой диапазон подстрок с номерами от M до N. Если последний символ в строке TOKENS= является звездочкой, то создается дополнительная переменная, значением которой будет весь текст, оставшийся в строке после обработки последней подстроки.

Разберем использование этой команды на примере пакетного файла parser.bat, который производит разбор файла myfile.txt:

```
@ECHO OFF
IF NOT EXIST myfile.txt GOTO :NoFile
FOR /F "EOL=; TOKENS=2,3* DELIMS=, " %%i IN (myfile.txt)
  DO @ECHO %%i %%j %%k
GOTO :EOF
:NoFile
ECHO Не найден файл myfile.txt!
```

Здесь во второй строке производится проверка наличия файла myfile.txt: в случае отсутствия этого файла выводится предупреждающее сообщение.

Команда FOR в третьей строке обрабатывает файл myfile.txt следующим образом:

1. Пропускаются все строки, которые начинаются с символа точки с запятой (EOL=;).
2. Вторая и третья подстроки из каждой строки передаются в тело цикла, причем подстроки разделяются пробелами (по умолчанию) и/или запятыми (DELIMS=,).
3. В теле цикла переменная %%i используется для второй подстроки, %%j — для третьей, а %%k получает все оставшиеся подстроки после третьей.

В нашем примере переменная %%i явно описана в инструкции FOR, а переменные %%j и %%k задаются неявно с помощью ключа TOKENS=. Например, если в файле myfile.txt были записаны следующие три строки:

```
ААА БББББ ВВВВ,ГГГГГ ДДДД
ЕЕЕЕЕ,ЖЖЖЖ 3333
;ККККК ЛЛЛЛЛ МММММ
```

то в результате выполнения пакетного файла parser.bat на экран выведется следующее:

```
ББББ ВВВВ ГГГГГ ДДДД
ЖЖЖЖ 3333
```

Замечание

Ключ TOKENS= позволяет извлечь из одной строки файла до 26 подстрок, поэтому запрещено использовать имена переменных, начинающиеся не с букв английского алфавита (a–z). Следует помнить, что имена переменных FOR являются глобальными, поэтому одновременно не может быть активно более 26 переменных.

Команда FOR /F также позволяет обработать отдельную строку. Для этого следует ввести нужную строку в кавычках вместо набора имен файлов в скобках. Строка будет обработана так, как будто она взята из файла. Например, файл следующего содержания:

```
@ECHO OFF
FOR /F "EOL=; TOKENS=2,3* DELIMS=, " %%i IN ("ААА БББББ ВВВВ,ГГГГГ ДДДД")
DO @ECHO %%i %%j %%k
```

при своем выполнении напечатает

```
ББББ ВВВВ ГГГГГ ДДДД
```

Вместо явного задания строки для разбора можно пользоваться переменными среды, например:

```
@ECHO OFF
SET M=ААА БББББ ВВВВ,ГГГГГ ДДДД
FOR /F "EOL=; TOKENS=2,3* DELIMS=, " %%i IN ("%M%") DO @ECHO %%i %%j %%k
```

Наконец, команда FOR /F позволяет обработать строку вывода другой команды. Для этого следует вместо набора имен файлов в скобках ввести строку вызова команды в апострофах (не в кавычках!). Строка передается для выполнения интерпретатору команд cmd.exe, а вывод этой команды записывается в память и обрабатывается так, как будто строка вывода взята из файла. Например, следующий командный файл:

```
@ECHO OFF
CLS
ECHO Имена переменных среды:
ECHO.
FOR /F "DELIMS==" %%i IN ('SET') DO ECHO %%i
```

выведет перечень имен всех переменных среды, определенных в настоящее время в системе.

В Windows NT также расширены операции подстановки ссылок на переменные команды FOR. В FOR допускается применение тех же синтаксических конструкций (операторов), что и для заменяемых параметров (*см. разд. "Переменные и параметры командной строки" ранее в этой главе*); описание этих операторов приведено в табл. 4.4.

Таблица 4.4. *Операторы для переменных команды* FOR

Операторы	Описание
%~Fi	Переменная %i расширяется до полного имени файла
%~Di	Из переменной %i выделяется только имя диска
%~Pi	Из переменной %i выделяется только путь к файлу
%~Ni	Из переменной %i выделяется только имя файла
%~Xi	Из переменной %i выделяется расширение имени файла
%~Si	Значение операторов N и X для переменной %i изменяется так, что они работают с кратким именем файла

Замечание

Если планируется использовать расширения подстановки значений в команде FOR, то следует внимательно подбирать имена переменных, чтобы они не пересекались с обозначениями формата.

Например, если мы находимся в каталоге C:\Program Files\Far и запустим командный файл следующего содержания:

```
@ECHO OFF
CLS
FOR %%i IN (*.txt) DO ECHO %%~Fi
```

то на экран выведутся полные имена всех файлов с расширением txt:

```
C:\Program Files\Far\Contacts.txt
C:\Program Files\Far\FarFAQ.txt
C:\Program Files\Far\Far_Site.txt
C:\Program Files\Far\License.txt
C:\Program Files\Far\License.xUSSR.txt
C:\Program Files\Far\ReadMe.txt
C:\Program Files\Far\register.txt
C:\Program Files\Far\WhatsNew.txt
```

Перейдем теперь к рассмотрению команды FORFILES из пакета Windows NT Resource Kit, которая предназначена для поиска файлов в одном или нескольких каталогах и выполнения над этими файлами каких-либо действий. FORFILES позволяет применять команды, предназначенные для обработки одного файла, к целому множеству файлов, а также передавать аргументы для набора исполняемых файлов. Используя FORFILES, можно, скажем, запустить команду TYPE для всех файлов с расширением txt в заданном каталоге или выполнить все командные файлы на диске C: с именем файла myinput.txt в качестве первого параметра.

Конечно, для подобных целей также служит стандартная команда FOR ... IN ... DO, однако FORFILES обладает более простым синтаксисом:

```
FORFILES [-Pпуть] [-Mмаска] [-Cкоманда] [-Dдата] [-S] [-V] [-?]
```

Замечание

Отметим, что в этой команде ключи указываются с помощью знака -, а не /. Встроенная справка выводится с помощью ключа -?.

Ключ -P*путь* указывает каталог, с которого следует начать поиск. По умолчанию таким начальным каталогом является тот, из которого запущена команда FORFILES.

Ключ -M*маска* задает шаблон для поиска файлов, например, *.txt. По умолчанию маска поиска задает все файлы (*.*).

Ключ -C*команда* определяет команду, которая будет выполняться для каждого найденного файла. В задаваемой для выполнения команде можно использовать три специальные переменные: @FILE, @PATH и @RELPATH. Вместо переменной @FILE подставляется имя найденного файла, вместо @PATH — полный путь к этому файлу, а вместо @RELPATH — относительный путь к найденному файлу из каталога, заданного ключом -P*путь* (или из текущего каталога, если ключ -P*путь* не указан).

Замечание

В именах переменных @FILE, @PATH и @RELPATH важен регистр символов.

Если вам нужно выполнить внутреннюю команду (например, COPY), то необходимо перед этой командой запустить копию интерпретатора cmd.exe, т е указать команду CMD /C Командой по умолчанию для FORFILES является следующая

```
"CMD /C ECHO @FILE"
```

Ключ *-Dдата* позволяет выделять только те файлы, дата которых больше или равна указанной дате Сам параметр *дата* задается в виде ДДММГГ, например, 300901

Ключ -S позволяет искать файлы во всех подкаталогах заданного каталога Приведем пример пакетного файла findbat.bat, печатающего список командных файлов с расширением bat во всех каталогах диска, указанного в качестве параметра командной строки (см листинг 4 3)

Листинг 4.3. Файл findbat.bat

```
@ECHO OFF
IF -%1==- GOTO :NoParam
ECHO Все командные файлы на диске %1·
FORFILES -P%1:\ -s -M*.BAT -C"CMD /C ECHO Файл· @PATH\@FILE"
ECHO ----------------------------------------------------
GOTO :EOF
:NoParam
ECHO Не задан диск для поиска файлов!
```

По умолчанию команда FORFILES в процессе работы выводит на экран только результат выполнения команды, заданной ключом -C При использовании ключа -V можно запустить FORFILES в режиме вывода развернутой информации При этом на экран будет выводиться сканируемый каталог, для каждого найденного файла будет печататься выполняемая команда, затем результат действия этой команды Кроме этого, для каждого каталога напечатается количество найденных файлов, а в самом конце выведется общее количество найденных файлов во всех проверенных каталогах

Команды *PUSHD* и *POPD*

Иногда бывает необходимо в командных файлах запоминать текущий каталог, переходить в другой каталог, выполнять какие-либо действия, а затем возвращаться в исходный Для этого в Windows NT включены две команды PUSHD и POPD

Команда PUSHD сохраняет имя текущего каталога для команды POPD и осуществляет переход в другой каталог Ее синтаксис имеет вид

```
PUSHD [путь|   ]
```

Здесь параметр *путь* задает каталог, который будет сделан текущим

Когда включена расширенная обработка команд, команда PUSHD допускает ввод сетевых путей в дополнение к обычным именам дисков и путям Например

```
PUSHD \\Server1\Programs
```

Если указан сетевой путь, команда PUSHD создает временное имя диска, указывающее на заданный сетевой ресурс, а затем производит смену текущего диска и каталога, используя вновь определенное имя диска Выделение временных имен дисков проводится в обратном порядке, начиная с Z., причем выбирается первое свободное имя диска

Вновь сделать текущим каталог, сохраненный командой PUSHD, можно с помощью команды POPD Команда POPD может быть использована только один раз для изменения каталога, после чего буфер будет очищен Когда включена расширенная обработка команд, команда POPD удаляет временные имена дисков, созданные PUSHD для сетевых ресурсов

Таким образом, PUSHD и POPD можно использовать в командных файлах для возврата в каталог, откуда была вызвана пакетная программа Приведем пример пакетного файла deltxt.bat, который удаляет все файлы с расширением txt в каталоге, заданном первым параметром командной строки (см листинг 4 4)

Листинг 4.4. Файл deltxt.bat

```
@ECHO OFF
IF -%1==- GOTO NoParam
REM Переходим в нужный каталог
PUSHD %1
DEL * txt
REM Возвращаемся в исходный каталог
POPD
CLS
ECHO Все текстовые файлы в каталоге %1 удалены'
GOTO .EOF
:NoParam
ECHO Не задано имя каталога'
PAUSE
```

Утилиты Windows NT Resource Kit

В этом разделе мы обсудим следующие команды из пакета Windows NT Resource Kit, которые могут быть применены в командных файлах· CHOICE, LOGTIME, NOW, TIMETHIS, SLEEP, TIMEOUT, FREEDISK

Как уже отмечалось выше, в Windows NT Resource Kit вновь включена утилита CHOICE, непонятно по каким причинам отсутствующая среди стандартных команд Windows NT. Напомним, что команда CHOICE выводит пользователю заданную подсказку и ждет, пока он выберет нужный вариант из указанного набора клавиш; подробное описание этой команды приведено в *разд. "Диалоговые командные файлы" гл. 2.*

Рассмотрим утилиты, позволяющие фиксировать и оценивать время и продолжительность выполнения заданных команд.

Начнем с команды LOGTIME, которая позволяет протоколировать время начала или завершения команды либо программы, вызванной из пакетного файла. Такая операция может быть полезна для отслеживания времени работы командных файлов. Ее синтаксис имеет вид:

```
LOGTIME [комментарий]
```

В результате своей работы команда LOGTIME создает файл-протокол с фиксированным именем logtime.log, в который записываются текущие дата и время, а также заданный текстовый комментарий. Если файл logtime.log уже существовал, информация будет дописываться в его конец.

Если, например, LOGTIME запускается до и после какой-либо команды в пакетном файле, то в файл-протокол запишется время начала и завершения работы этой команды:

```
LOGTIME "Начало процедуры импорта"
import.exe
LOGTIME "Завершение процедуры импорта"
```

В этом случае в файл logtime.log запишется информация следующего вида:

```
04/10/01 13:14:19 Начало процедуры импорта
04/10/01 13:15:07 Завершение процедуры импорта
```

Недостатком команды LOGTIME является невозможность изменить файл, в который производится вывод.

Похожей на LOGTIME командой является утилита NOW, которая выводит в стандартный выходной поток STDOUT текущую дату, время и текстовое сообщение. Синтаксис этой команды имеет вид:

```
NOW [сообщение]
```

Команда NOW похожа на утилиту ECHO, но с дополнительной информацией о текущей дате и времени. Например, команда

```
NOW Копирование завершено
```

выведет на экран сообщение следующего вида:

```
Tue Apr 10 10:57:26 2001 Копирование завершено
```

В случае необходимости, вывод команды NOW может быть перенаправлен в файл или другое устройство с помощью символов перенаправления > или >>. Это позволяет формировать файлы-протоколы с произвольными именами. Например, для заполнения файла C:\import.log в пакетном файле должны иметься, например, такие строки:

```
NOW "Начало процедуры импорта" >> import.log
import.exe
NOW " Завершение процедуры импорта" >> import.log
```

Еще одной командой, связанной с протоколированием времени выполнения программ, является TIMETHIS. Эта утилита позволяет определить, сколько времени требуется системе на выполнение той или иной команды или программы. Ее синтаксис имеет вид:

```
TIMETHIS команда
```

Здесь в качестве параметра указывается имя команды или программы, которую вы хотите запустить и оценить время ее выполнения. В случае необходимости для запускаемой команды можно указать дополнительные параметры и ключи. Например, выполнив следующую команду:

```
TIMETHIS DIR C:\USERS\DEFAULT\TEMP
```

мы получим на экране информацию такого вида:

```
TimeThis :  Command Line :  DIR C:\USERS\DEFAULT\TEMP
TimeThis :    Start Time :  Tue Apr 10 11:23:22 2001
   Volume in drive C has no label.
   Volume Serial Number is 8C25-C783
   Folder of C:\USERS\DEFAULT\TEMP
03/08/00  11:04         <DIR>          .
03/08/00  11:04         <DIR>          ..
20/03/00  12:31                54,717 BIGLMH
20/03/00  14:57                55,801 HOSTS
20/03/00  14:57                55,801 LMHOSTS
17/03/00  05:10                24,274 LMHOSTS.000
               6 File(s)        190,593 bytes
                              39,556,096 bytes free
TimeThis :  Command Line :  DIR C:\USERS\DEFAULT\TEMP
TimeThis :    Start Time :  Tue Apr 10 11:23:22 1995
TimeThis :      End Time :  Tue Apr 10 11:23:23 1995
TimeThis :  Elapsed Time :  00:00:00.972
```

Таким образом, TIMETHIS выполняет заданную команду (в нашем примере это DIR C:\USERS\DEFAULT\TEMP), затем сообщает время (в формате ЧЧ:ММ:СС.ТТТ, где ТТТ — тысячные доли секунды) запуска (TimeThis: Start Time: ...) и завершения команды (TimeThis: End Time: ...), а также продолжительность ее выполнения (TimeThis: Elapsed Time: ...).

Информацию, выводимую командой TIMETHIS, можно направить в файл Дл этого, как обычно, используются символы перенаправления > и >>, на пример·

```
TIMETHIS DIR C:\USERS\DEFAULT\TEMP > dir.out
```

Если же символы перенаправления ввода/вывода входят в саму оцениваемую команду, то такую команду нужно заключать в кавычки Например

```
TIMETHIS "DIR > dirlist.txt"
```

Это делается для того, чтобы отличить перенаправление, заданное выполняемой командой, от перенаправления, которое может использовать сама утилита TIMETHIS

Следующие две утилиты, SLEEP и TIMEOUT, позволяют приостановить выполнение пакетного файла на определенное время Другими словами, эти команды задают период бездействия

Синтаксис для команды SLEEP имеет вид

```
SLEEP n
```

Здесь параметр n задает длительность паузы в секундах Например, команда

```
SLEEP 3600
```

приостановит выполнение командного файла на один час, после чего выполнится команда, следующая в файле за SLEEP

Рассмотрим пример пакетного файла, который используется в качестве сценария регистрации (подобные сценарии будут описаны в следующем разделе) для выдачи информации пользователям при их регистрации в сети

```
@ECHO OFF
ECHO * Сообщение для всех !!! *
ECHO.
ECHO Сегодня, 20 августа, состоится общее собрание.
ECHO Начало в 16-00.
SLEEP 60
```

Когда при регистрации пользователя выполнится этот файл, заданные строки сообщения

```
* Сообщение для всех !!! *

Сегодня, 20 августа, состоится общее собрание.
Начало в 16-00.
```

будут находиться на экране в течение минуты, после чего командное окно автоматически закроется

Похожей на SLEEP командой является TIMEOUT, которая также приостанавливает работу командного интерпретатора на заданное число секунд

```
TIMEOUT n
```

Здесь, как и в команде SLEEP, параметр n задает период бездействия (от −1 до 100 000) в секундах Значение −1 означает, что система будет неограниченно долго ждать нажатия на клавишу (в этом случае TIMEOUT становится идентичной команде PAUSE)

Отличием команды TIMEOUT от SLEEP является то, что пользователь может в любой момент прервать бездействие, нажав любую клавишу Таким образом, TIMEOUT объединяет в себе свойства команд PAUSE и SLEEP

С помощью утилиты FREEDISK можно проверить наличие свободного места на диске (жестком или гибком) Синтаксис этой команды имеет вид

```
FREEDISK диск объем
```

Параметр *диск* здесь задает имя диска, для которого проводится проверка (A:, B:, C:, ...), а объем запрашивает количество свободного места (в байтах) на этом диске Если на диске места достаточно, то на экран выводится **OK** и устанавливается код возврата ERRORLEVEL, равный нулю В противном случае печатается Too small!! и код возврата становится равным единице Это свойство можно использовать в командных файлах (см листинг 4 5)

Листинг 4.5. Командный файл, проверяющий перед копированием наличие свободного места на дискете

```
@ECHO OFF
CLS
ECHO Вставьте дискету в дисковод A:
PAUSE
REM Проверка наличия 400000 свободных байтов на диске A:
FREEDISK A: 400000
IF ERRORLEVEL 1 GOTO NoSpace
XCOPY C:\ADITOR A:\
GOTO :EOF
:NoSpace
ECHO На диске A: недостаточно места!
PAUSE
```

Сценарии входа для пользователей

Напомним, что в Windows NT для настройки среды пользователя используются *профили* (локальные и серверные), в состав которых входят все настраиваемые пользователем параметры язык и региональные настройки,

настройка мыши и звуковых сигналов, подключаемые сетевые диски и принтеры и т. д. Профили, сохраняемые на сервере, обеспечивают пользователям одну и ту же рабочую среду вне зависимости от того, с какого компьютера (под управлением Windows NT) зарегистрировался пользователь. Создание и поддержание профилей пользователей описываются практически в любой книге по администрированию Windows NT и здесь рассматриваться не будут.

Иногда для настройки среды пользователей вместо профилей применяются *сценарии входа* — командные или исполняемые файлы, которые запускаются на машине пользователя каждый раз при его регистрации в сети. Перечислим несколько причин применения сценариев входа в дополнение или вместо профилей.

- На сервере регистрируются пользователи, работающие на компьютерах с различными операционными системами (Windows NT, Windows 9*x*, MS-DOS, OS/2). Профили в полном объеме работают только на рабочих станциях под Windows NT.
- Администратор должен задавать только некоторые параметры среды пользователя, не вмешиваясь в остальные настройки.
- Сценарии входа легче создавать и поддерживать, чем профили.

Сценарии входа выполняются на компьютере пользователя, а хранятся в едином каталоге (по умолчанию в `%SystemRoot%\System32\Repl\Import\Scripts`) на сервере — основном контроллере домена, что довольно удобно с точки зрения администратора. Таким образом, когда пользователь регистрируется, компьютер, производящий аутентификацию, ищет сценарий входа в этом специальном каталоге на сервере.

Для того чтобы стандартными средствами назначить сценарий входа определенному пользователю, необходимо запустить **Диспетчер пользователей** (User Manager for Domain), выбрать из списка нужного пользователя и в окне **Свойства пользователя** (User Properties) нажать на кнопку **Профиль** (Profile); в результате на экран выведется показанное на рис. 4.3 диалоговое окно **Профиль пользователя** (User Environment Profile). Имя сценария входа вводится в поле **Сценарий входа** (Logon Script) этого окна.

Также можно создавать в каталоге `\Script` подкаталоги для помещения туда определенных сценариев. Так как один сценарий входа может быть назначен нескольким пользователям, то таким образом можно сгруппировать пользователей в группы и назначить соответствующие сценарии входа каждой группе. Например, такой общий сценарий можно создать для пользователей, работающих в определенной операционной системе.

Для того чтобы задать путь к сценарию входа, который находится в подкаталоге каталога `\Script`, необходимо в поле **Сценарий входа** (Logon Script) указать относительный путь к этому файлу из каталога `%SystemRoot%\System32\Repl\Import\Scripts`. Если, скажем, сценарий `scr99.bat` для пол-

зователя Kazakov находится в каталоге с полным именем C:\Winnt \System32\Repl\Import\Scripts\Script99, то в качестве пути к сценарию входа нужно указывать \Script99\scr99.bat.

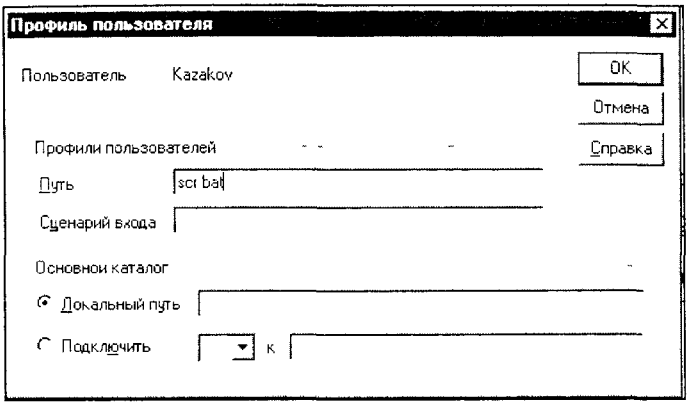


Рис. 4.3. Настройки профиля пользователя в Windows NT Workstation 4.0

Замечание

Отметим, что весь каталог %SystemRoot%\System32\Repl гарантированно реплицируется (синхронизируется) по всем контроллерам домена, поэтому все сценарии входа будут доступны вне зависимости от того, какой из контроллеров домена выполняет аутентификацию на момент регистрации пользователя.

Как уже отмечалось, сценарий входа может быть командным файлом (с расширением bat или cmd), либо выполняемой программой (с расширением exe). Расширение имени файла определяется конкретной операционной системой, под управлением которой работает компьютер пользователя:

- клиенты MS-DOS, Windows for Workgroups, Windows 9*x* и Windows NT 3.1 должны использовать для файлов со сценариями входа расширение bat;
- для клиентов OS/2 следует использовать расширение cmd;
- клиенты Windows NT 4.0 (Server и Workstation) могут использовать оба расширения (bat и cmd).

Замечание

В сценарии входа должны применяться только команды, поддерживаемые операционной системой на компьютере пользователя. Скажем, если на сервере регистрируется пользователь, работающий под Windows 98, а в сценарии входа для этого пользователя присутствуют команды, специфические для Windows NT (например, SETLOCAL), то эти команды выполнены не будут.

Отметим также, что если сценарий входа находится на сервере с файловой системой NTFS, то необходимо предоставить тем пользователям, для которых

предназначен этот сценарий, право на его чтение (Read). В противном слу чае при регистрации пользователя в сети нужный сценарий просто не за пустится.

Чаще всего сценарии входа используются для подключения дисков и портов принтера к сетевым ресурсам, а также для синхронизации системного вре мени пользовательских компьютеров с системным временем сервера (это необходимо, например, для файл-серверных бухгалтерских систем, рабо тающих в реальном времени). Напомним, что подключить сетевые ресурсы можно с помощью команда `NET USE`, а для синхронизации времени предна значена команда `NET TIME`. Обе эти команды подробно описаны в *разд "Команды для работы с локальной сетью" гл. 1.* Например, для пользователя работающего с Windows NT, сценарий входа может иметь следующий вид:

```
@ECHO OFF
NET TIME \\Server1 /SET /YES
NET USE LPT2: \\Server1\Epson /PERSISTENT:NO
NET USE E: \\Server1\Documents /PERSISTENT:NO
```

Здесь производится синхронизация системного времени рабочей станции временем сервера `Server1`, порт `LPT2` подключается к сетевому принтер `\\Server1\Epson`, а диск `E:` — к сетевому ресурсу `\\Server1\Documents`.

Ключ `/PERSISTENT:NO` в командах `NET USE` нужен для создания временны подключений (не сохраняющихся после завершения сеанса пользователя Если бы подключения были постоянными `/PERSISTENT:YES`, то при следую щем входе пользователя в систему возникла бы ошибка (повторное испол зование уже имеющегося подключения).

Замечание

Напомним, что у команды `NET USE` в Windows 9x такого ключа нет, там все по ключения являются временными.

Для написания универсальных сценариев входа для пользователей, раб тающих с Windows NT, бывает полезно использовать значения переменны среды, приведенных в табл. 4.5.

Таблица 4.5. *Переменные среды, полезные для использовани в сценариях входа*

Переменная	Описание
`%HOMEDIR%`	Буква переопределенного диска на компьютере пользователя, которая ссылается на сетевой ре сурс, содержащий личный каталог пользователя

Таблица 4.5 *(окончание)*

Переменная	Описание
%HOMEDRIVE%	Локальный, либо перенаправленный диск, на котором расположен личный каталог
%HOMEPATH%	Путь к личному каталогу
%HOMESHARE%	Имя каталога общего доступа, включающее личный каталог и локальный, либо переопределенный диск
%OS%	Операционная система, управляющая рабочей станцией
%PROCESSOR_ARCHITECTURE%	Архитектура процессора (например, x86) рабочей станции пользователя
%PROCESSOR_LEVEL%	Тип процессора (например, 486) рабочей станции пользователя
%USERDOMAIN%	Домен, в котором зарегистрирован пользователь
%USERNAME%	Имя, под которым регистрировался при входе в сеть пользователь

В сценариях входа очень полезной является команда IFMEMBER из пакета Windows NT Resource Kit, которая позволяет проверять принадлежность пользователя, выполняющего регистрацию, к определенной группе (например, администраторов).

Синтаксис этой команды имеет вид:

```
IFMEMBER [группа1] [группа2] ... [группаN]
```

Параметры *группа1*, *группа2*, ..., *группаN* задают список групп, принадлежность к которым нужно проверить для регистрирующегося пользователя. Код завершения команды IFMEMBER будет равен номеру группы, в которую входит пользователь, как параметра командной строки. Для анализа этого номера в пакетном файле применяется команда IF ERRORLEVEL. Следующий сценарий входа выведет информацию на экран, только если пользователь принадлежит к группе администраторов:

```
@ECHO OFF
IFMEMBER Administrators
IF NOT ERRORLEVEL 1 EXIT
ECHO Поздравляю, %USERNAME%, Вы — Администратор!
```

Заключение

В командных файлах Windows NT появляется довольно много новых воз- можностей. Перечислим наиболее важные из них.

- Над переменными среды, заменяемыми параметрами и переменными цикла, можно выполнять арифметические действия, а также проводить определенный синтаксический анализ.
- Изменения переменных среды можно локализовать внутри командного файла.
- Существует возможность немедленного перехода в конец пакетного файла.
- Имеется механизм перехода к заданной метке в новой копии текущего пакетного файла (с произвольными параметрами командной строки).
- Усовершенствована работа условного оператора `IF` (в частности, разре- шается использовать операторы условия, отличные от сравнения на ра- венство).
- Появилось несколько новых видов циклов. В частности, реализована возможность чтения и обработки строк из текстового файла, а также об- работки строк вывода определенной команды.
- Команды из пакета Windows NT Resource Kit позволяют приостанавли- вать выполнение командного файла на определенное время, а также со- давать текстовые файлы-протоколы производимых действий.

Однако при всех своих усовершенствованиях пакетные файлы все-же нель использовать как полноценный инструмент для администрирования опер ционной системы Windows. Упомянем лишь наиболее бросающиеся в гла недостатки пакетных файлов.

- Отсутствует полноценный оператор `IF ... ELSE`.
- Нет прямых инструкций, позволяющих читать и записывать текстовые файлы.
- Нельзя напрямую работать с рабочим столом Windows и другими специ- альными папками.

Для написания полноценных сценариев, работающих под управлением Windows, нужно использовать разработанный фирмой Microsoft сервер сце- нариев Windows (Windows Scripting Host, WSH), описанию которого посвя- щены следующие две главы.

Упражнения

Для проверки знаний, полученных при прочтении настоящей главы, читателям предлагается выполнить следующие упражнения.

1. Написать командный файл, который печатал бы общее число переменных среды, определенных в системе, и после нажатия клавиши выводил на экран имена этих переменных (без значений) вместе с порядковым номером. Таким образом, на экран должна выводиться информация следующего вида:

```
Количество переменных в системе: 33
-----------------------------------
1. CLASSPATH
2. CLIPPER
3. COMPUTERNAME
4. COMSPEC
5. HOMEDRIVE
6. HOMEPATH
...
```

2. Усложнение предыдущего упражнения. На экране печатать только число переменных среды, а всю остальную информацию выводить в заданный в качестве параметра командной строки текстовый файл, который затем открыть в Блокноте (Notepad). Если файл для вывода не задан, то выводить список на экран. Также предусмотреть два дополнительных ключа: если задан ключ /B, то выводить только имена переменных (без значений, как в предыдущем упражнении), если задан ключ /A, то выводить имя переменной и ее значение в круглых скобках:

```
1. CLASSPATH    (C:\PVSW\BIN\psql.jar)
2. CLIPPER    (f80)
3. COMPUTERNAME    (404_POPOVNT)
4. ComSpec    (C:\WINNT\system32\cmd.exe)
5. HOMEDRIVE    (C:)
6. HOMEPATH    (\)
...
```

   По умолчанию считать заданным ключ /B. Если пакетный файл запускается вообще без параметров, то вывести описание его синтаксиса:

```
listvar.bat [/A|/B] [file_name]
```

3. Написать пакетный файл, который автоматически удалял бы в каталогах D:\Profiles и D:\HomeDirs все подкаталоги, размер которых превышает 20 Мбайт.

4. Создать сценарий входа, который считывал бы персональное сообщение для пользователя из текстового файла, находящегося на общем сетевом ресурсе, выводил это сообщение на экран и оставлял его там в течение 30 секунд, после чего закрывал командное окно. Если сообщения для данного пользователя в файле нет, ничего на экран выводить не нужно. Пример файла с сообщениями:

```
ivanov Зайдите срочно в бухгалтерию!
petrov Будьте сегодня на рабочем месте в 15-00
sidorov Здравствуйте, товарищ Сидоров!
```

5. Написать сценарий входа, который выполнял бы следующие действия:
   - синхронизировал системное время клиентского компьютера с системным временем на сервере `Server1`;
   - подключал диск `M:` к сетевому ресурсу `\\Server1\Letters`;
   - предоставлял каталог `C:\TEXT` на клиентском компьютере в общее пользование (только чтение).
6. Пусть имеется текстовый файл `sums.txt` с разделителями следующего формата:

```
Фамилия|Имя|Отчество|Сумма
```

   Например:

```
Петров|Петр|Петрович|1450
Иванов|Иван|Иванович|1200
Сидоров|Андрей|Николаевич|1123
...
```

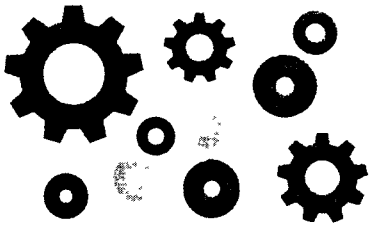
   Необходимо написать пакетный файл `seeksum.cmd`, который запускался бы с двумя параметрами командной строки:

```
seeksum.cmd MIN MAX
```

   где *MIN* — минимальная сумма, *MAX* — максимальная сумма, и искал в файле `sums.txt` всех людей, у которых сумма меньше либо равна *MAX*, но больше либо равна *MIN*. Информацию выводить в файл `suminfo.txt`, причем фамилии должны идти в алфавитном порядке. Например, если `seeksum.cmd` запущен с параметрами `1150` и `1500`, то файл `suminfo.txt` должен иметь следующий вид:

```
Диапазон сумм: от 1150 до 1500
Иванов = 1200
Петров = 1450
...
```

ГЛАВА 5



Введение в Windows Scripting Host (WSH)

Рассмотренные нами в первых четырех главах командные файлы (особенно те из них, которые используют утилиты пакета Windows NT Resource Kit и расширенный режим командного процессора `cmd.exe` в Windows NT) являются очень удобным средством для надежной и быстрой автоматизации несложных повторяющихся задач, не требующих организации интерактивного диалога с пользователем. Однако язык командных файлов никак нельзя назвать полноценным языком программирования, и более-менее сложные сценарии с помощью этого языка написать весьма проблематично, а порой и невозможно (недостатки командных файлов были перечислены в заключении к гл. 4). Для решения этой проблемы несколько компаний в разное время предлагали разработанные ими языки сценариев для Windows и интерпретаторы этих языков (CEnvi фирмы Nombas, Power Script фирмы Desiderate Software, WinBatch фирмы Wilson WindowWare Inc. и т. д.), но в силу различных причин широкого распространения эти средства не получили. В настоящее время в качестве инструмента для написания гибких и мощных сценариев для Windows широко используется разработанный фирмой Microsoft сервер сценариев Windows Scripting Host (WSH), рассмотрению возможностей которого и будут посвящены *гл. 5* и *6*.

Принцип работы сценариев, которые выполняются с помощью WSH, состоит в использовании *объектов ActiveX*, поэтому вначале мы очень кратко опишем возможности самой технологии ActiveX компании Microsoft.

Возможности технологии ActiveX

Напомним, что в Windows с самого начала для обеспечения обмена данными между приложениями была разработана технология *связывания и внедрения объектов* (Object Linking and Embedding, OLE). Сначала технология OLE использовалась для создания составных документов, а затем для решения более общей задачи — предоставления приложениями друг другу собствен-

ных функций (служб) и правильного использования этих функций. Технология, позволяющая одному приложению *(клиенту автоматизации)* вызывать функции другого приложения *(сервера автоматизации)* была названа OLE Automation. В основе OLE и OLE Automation лежит разработанная Microsoft базовая "компонентная" технология Component Object Model (COM). В общих словах, компонентное программное обеспечение — это способ разработки программ, при котором используются технологии создания программных модулей, подобные технологиям, применяемым для разработки аппаратных средств. Сложные элементные схемы собираются из стандартизированных микросхем, которые имеют четко определенные документированные функции. Разработчик может эффективно пользоваться такими микросхемами, не задумываясь об их внутренней структуре. В программных компонентах, написанных на каком-либо языке программирования, детали реализации используемых алгоритмов также скрыты внутри компонента (объекта), а на поверхности находятся общедоступные *интерфейсы*, которыми могут пользоваться и другие приложения, написанные на том же или другом языке.

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

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

Сервер сценариев WSH является мощным инструментом, предоставляющим единый интерфейс для специализированных языков (в том числе для VBScript и JScript), которые, в свою очередь, позволяют использовать любые внешние объекты ActiveX. С помощью WSH сценарии могут быть выполнены непосредственно в операционной системе Windows.

Назначение и основные свойства WSH

При помощи WSH можно выполнять сценарии, написанные на любом языке, для которого установлен соответствующий модуль (scripting engine), поддерживающий технологию ActiveX Scripting. Сценарии могут запускаться несколькими способами: с рабочего стола или из Проводника Windows (с помощью исполняемого файла `wscript.exe`), и из командной строки (с помощью файла `cscript.exe`).

Используя сценарии WSH, можно непосредственно работать с файловой системой компьютера, а таже управлять работой других приложений (серверов автоматизации). При этом возможности сценариев ограничены только средствами, которые предоставляют доступные серверы автоматизации.

WSH предъявляет минимальные требования к объему оперативной памяти и является очень удобным инструментом для написания сложных сценариев регистрации пользователей и автоматизации повседневных задач.

В поставку Windows 98 входит WSH версии 1.0, а в Windows 2000 — версии 2.0. Кроме этого, последнюю версию WSH можно свободно скачать с сервера Microsoft (**http://msdn.microsoft.com/scripting**).

Мы будем рассматривать WSH версии 2.0, в которой Microsoft реализовала поддержку двух языков сценариев: VBScript и JScript. В принципе, можно пользоваться и разработками сторонних компаний, которые предоставляют свои модули для других языков, например, PerlScript, TCL, REXX и Python.

По своим функциональным возможностям языки VBScript и JScript в настоящее время стали практически равнозначны, однако внешне их синтаксис различен: VBScript основан на языке Visual Basic, а JScript похож на Java и C. Надо отметить, что большинство примеров сценариев WSH, которые встречаются в литературе и на сайтах сети Internet, написаны на VBScript. Напротив, все приведенные далее в книге примеры сценариев написаны с помощью языка JScript (это объясняется, в большей степени, личными пристрастиями автора); в случае необходимости, перевод предложенных сценариев на VBScript не представляет никакой сложности. Отметим также, что WSH 2.0 позволяет в одном сценарии использовать языки JScript и VBScript одновременно (подробнее это описано в *разд. "Сценарии WSH как приложения XML" гл. 6*).

Описанию языка JScript посвящено большое количество доступной литературы, однако практически везде этот язык рассматривается с точки зрения управления браузером и создания интерактивных HTML-страниц. Мы же в следующем разделе рассмотрим этот язык лишь в том объеме, в котором он может понадобиться при написании сценариев WSH.

WSH-сценарий, написанный на языке JScript, — это обычный текстовый файл с расширением `js`, создавать 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. Как уже отмечалось выше, для запуска написанного сценария существует несколько способов. Можно выполнить сценарий из командной строки с помощью консольной версии WSH, `cscript.exe`. Например, чтобы запустить сценарий, записанный в файле `my.js`, нужно выполнить команду

```
cscript my.js
```

В этом случае исполнение сценария контролируется с помощью параметров командной строки для `cscript.exe` (табл. 5.1), которые включают или отключают различные опции WSH (все эти параметры начинаются с символов `//`).

Таблица 5.1. *Параметры командной строки* `cscript.exe`

Параметр	Описание
`//I`	Выключает пакетный режим (по умолчанию). При этом на экран будут выводиться все сообщения об ошибках в сценарии
`//B`	Включает пакетный режим. При этом сообщения об ошибках в сценарии на экран выводиться не будут
`//T:nn`	Задает тайм-аут в секундах, т. е. сценарий будет выполняться `nn` секунд, после чего процесс прервется. По умолчанию время выполнения не ограничено
`//logo`	Выводит (по умолчанию) перед выполнением сценария информацию о версии и разработчике WSH
`//nologo`	Подавляет вывод информации о версии и разработчике WSH
`//H:CScipt` или `//H:WScript`	Делает `cscript.exe` или `wscript.exe` приложением для запуска сценариев по умолчанию. Если эти параметры не указаны, то по умолчанию подразумевается `wscript.exe`
`//S`	Сохраняет установки командной строки для текущего пользователя
`//?`	Выводит встроенную подсказку для параметров командной строки
`//E:engine`	Выполняет сценарий с помощью модуля, заданного параметром `engine`
`//D`	Включает отладчик
`//X`	Выполняет программу в отладчике
`//Job:<JobID>`	Запускает задание с индексом `JobID` из многозадачного `WS`-файла (структура `WS`-файлов будет описана в *разд. "Сценарии WSH как приложения XML" гл. 6*)

Например, команда

```
cscript //B my.js /a /b
```

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

Также сценарий можно выполнять с помощью (оконной) графической версии WSH, `wscript.exe`. Для этого следует из Проводника Windows дважды щелкнуть по иконке сценария, либо воспользоваться пунктом **Выполнить** меню **Пуск**, написав полное имя сценария в поле **Открыть** (рис. 5.1).

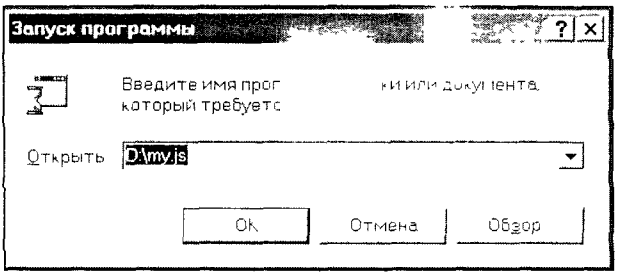


Рис. 5.1. Запуск сценария из меню **Пуск**

В этом случае вывод информации из сценария будет вестись в диалоговые окна Windows.

При использовании `wscript.exe` свойства сценария можно устанавливать с помощью вкладки **Script** диалогового окна, задающего свойства файла в Windows (рис. 5.2).

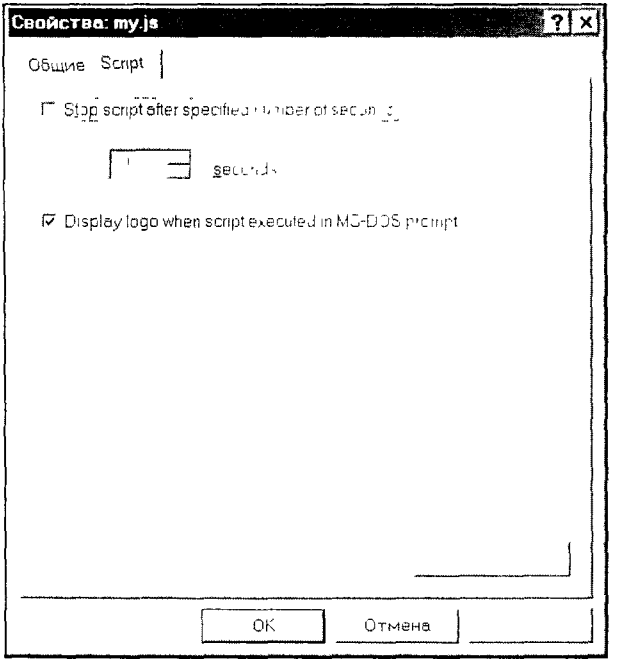


Рис. 5.2. Установка свойств сценария

После задания свойств сценария автоматически создается файл с именем этого сценария и расширением wsh, который имеет структуру наподобие ini-файла, например:

```
[ScriptFile]
Path=D:\my.js
[Options]
Timeout=0
DisplayLogo=1
```

Необходимые сведения о языке JScript

Язык JScript — это разработанный Microsoft интерпретируемый объектно-ориентированный язык сценариев, который первоначально предназначался для создания динамических HTML-страниц. Отметим, что JScript не является урезанной версией какого-либо другого языка программирования, хотя по синтаксису он похож на языки Java и C. Как уже было сказано выше, в этом разделе мы кратко рассмотрим те возможности и свойства JScript, которые могут потребоваться при составлении сценариев, выполняемых с помощью WSH.

Переменные

В сценариях JScript, как и в любом другом языке программирования, можно использовать переменные, обращаясь к ним по имени. При этом переменные могут быть как глобальными (доступными из любого места сценария), так и локальными (область действия ограничивается функцией, в которой они определены).

Хорошим тоном считается предварительное объявление используемых переменных с помощью ключевого слова var, хотя это является обязательным условием только для локальных переменных, определенных в функциях. Пример объявления переменной имеет вид:

```
var MyVariable;
```

При объявлении тип переменной явным образом не указывается (как это делается, например, в языках C или Pascal). Определенный тип переменной присваивается только тогда, когда в нее записывается какое-либо значение.

Язык JScript является регистро-зависимым, т. е. имена MyVariable и myvariable представляют разные переменные. Кроме этого, при выборе имен переменных следует придерживаться следующих правил:

- имя переменной должно начинаться с буквы или с символов "_", "$" и может состоять только из букв, цифр, а также символов "_", "$";

- имя переменной не должно совпадать с зарезервированными ключевыми словами языка JScript.

Список ключевых слов JScript приведен в табл. 5.2.

Таблица 5.2. *Зарезервированные ключевые слова JScript*

Break	default	false	new	true
Case	delete	finally	null	try
Catch	do	for	return	typeof
Class	else	function	super	var
Const	enum	if	switch	void
Continue	export	import	this	while
Debugger	extends	in	throw	with

Значения переменным в JScript присваиваются с помощью оператора присваивания "=". Например:

```
var MyVariable;
MyVariable = "Привет!";
```

Здесь мы объявили переменную `MyVariable` и записали в нее текстовую строку. Однако далее в любом месте сценария мы можем присвоить переменной `MyVariable` числовое значение (при этом тип переменной изменится), например:

```
MyVariable = 10;
```

Кроме этого, переменной можно присвоить специальное значение `null`:

```
MyVariable = null;
```

В данном случае переменной `MyVariable` не назначается никакого определенного типа (пустой тип). Такое присвоение применяется в тех случаях, когда необходимо объявить переменную и проинициализировать ее, не присваивая этой переменной никакого определенного типа и значения.

Типы данных

В JScript поддерживаются шесть типов данных, главными из которых являются числа, строки, объекты и логические данные. Оставшиеся два типа — это `null` (пустой тип) и `undefined` (неопределенный тип).

Числа

В сценариях JScript числа могут использоваться в различных форматах:

- целые числа (по основанию 8, 10 или 16);
- числа с плавающей десятичной точкой;
- числа в научной нотации.

Также имеются несколько специальных числовых значений:

- NaN (Not a Number) — так называемое "нечисло", которое не соответствует никакому числу (это значение генерируется в тех случаях, когда результат выполнения операции над числами не может быть представлен в виде числа);
- положительная бесконечность;
- отрицательная бесконечность;
- положительный 0;
- отрицательный 0.

Текстовые строки

Текстовые строки — это последовательность символов, заключенных в одинарные или двойные кавычки, например:

```
"Привет!"
'Большой привет!'
'И тогда он крикнул:"Берегись!"'
```

Строка может иметь нулевую длину (пустая строка):

```
MyVariable = "";
```

В JScript можно также использовать специальные комбинации символов, с помощью которых в строки включаются некоторые неотображаемые символы или символы, имеющие специальное значение. Каждая из этих комбинаций (escape-последовательностей) начинается с символа обратной косой черты "\" (табл. 5.3).

Таблица 5.3. *Специальные комбинации символов*

Escape-последовательность	Описание
\b	Backspace (забой)
\f	Перевод формата
\n	Перевод строки

Таблица 5.3 (окончание)

Escape-последовательность	Описание
\r	Возврат каретки
\t	Горизонтальная табуляция (<Ctrl>+<I>)
\'	Одиночная кавычка
\"	Двойная кавычка
\\	Обратная косая черта

Объекты

В JScript под объектом понимается совокупность *свойств* и *методов*. Метод — это внутренняя функция объекта, свойство — это одно значение какого-либо типа или несколько таких значений (в виде массива или объекта), хранящихся внутри объекта. Поддерживаются три вида объектов:

- встроенные (внутренние) объекты;
- объекты, создаваемые программистом в сценарии;
- внешние объекты (например, объекты WSH).

Более подробно объекты будут рассмотрены ниже.

Логические данные

Логические данные предназначены для выполнения операций сравнения, а также для использования в условных операторах. При этом логические данные имеют только два значения: `true` (истина) и `false` (ложь). Отметим, что в JScript эти значения никак не соотносятся с числами 1 и 0.

Null (пустой тип) и *undefined* (неопределенный тип)

Если переменная была объявлена с помощью ключевого слова `var`, но ей еще ни разу не присваивалось значение, она имеет неопределенный тип (`undefined`):

```
var MyVariable;
```

После выполнения этой строки переменная `MyVariable` имеет тип `undefined`. Как уже отмечалось выше, если теперь присвоить переменной значение `null`, то эта переменная будет типа `null` (пустой тип):

```
MyVariable = null;
```

Преобразование типов данных

Одной из особенностей языка JScript является то, что если в выражениях встречаются переменные разных типов, то автоматически происходит преобразование всех числовых данных в строковое представление. Например следующие логические выражения будут равны `true`:

```
"100" == 100
false == 0
```

(здесь `==` означает оператор сравнения). Для преобразования строк в числ нужно применять две специальные функции: `parseInt` (преобразование к целому числу) и `parseFloat` (преобразование к числу с плавающей запятой). Например, после выполнения следующих строк:

```
var s="";
s=(parseInt("3")-2)+"3";
```

значением переменной `s` будет строка `13`.

Операторы

В JScript поддерживаются операторы различных типов, которые похожи на операторы языка C.

Унарные операторы

Унарными называются операторы, которые применяются к одному операнду (табл. 5.4).

Таблица 5.4. *Унарные операторы*

Оператор	Описание
-	Изменение знака на противоположный
!	Дополнение. Используется для изменения значения логической переменной на противоположное
++	Увеличение значения числовой переменной на единицу (инкремент). Может применяться как префикс переменной или как ее суффикс
--	Уменьшение значения числовой переменной на единицу (декремент). Может применяться как префикс переменной или как ее суффикс

Бинарные операторы

Бинарными называются операторы, которые соединяют два операнда (табл. 5.5).

Таблица 5.5. *Бинарные операторы*

Оператор	Описание
-	Вычитание
+	Сложение
*	Умножение
/	Деление
%	Вычисление остатка от деления

Операторы побитовых логических операций и сдвига

Эти операторы позволяют производить над числовыми переменными побитовые операции, описанные в табл. 5.6.

Таблица 5.6. *Операторы побитовых логических операций и сдвига*

Оператор	Описание
&	Логическое И
|	Логическое ИЛИ
^	Логическое ИСКЛЮЧАЮЩЕЕ ИЛИ
~	Логическое НЕ
>>	Сдвиг вправо
<<	Сдвиг влево
>>>	Сдвиг вправо с заполнением освобождаемых разрядов нулями

Замечание

Перед использованием операторов из табл. 5.6 значения переменных преобразуются в 32-разрядные целые числа.

Операторы присваивания

В JScript, как и в языке C, для изменения содержимого переменных можно комбинировать оператор присваивания "=" с другими операторами (табл. 5.7).

Таблица 5.7. *Комбинации оператора присваивания и других операторов*

Оператор	Описание
=	Простое присваивание
+=	Увеличение численного значения или конкатенация (склеивание) строк
-=	Уменьшение численного значения
*=	Умножение
/=	Деление
%=	Вычисление остатка от деления
>>=	Сдвиг вправо
>>>=	Сдвиг вправо с освобождением освобождаемых разрядов нулями
<<=	Сдвиг влево
|=	Логическое ИЛИ
&=	Логическое И
^=	Логическое ИСКЛЮЧАЮЩЕЕ ИЛИ

Операторы отношения

Операторы отношения используются для сравнения значений нескольких переменных. Эти операторы, описанные в табл. 5.8, могут возвращать только логические значения `true` или `false`.

Таблица 5.8. *Операторы отношения*

Оператор	Условие, при котором возвращается значение `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

Также в условных операторах применяются логические операторы (табл. 5.9).

Таблица 5.9. *Логические операторы*

Оператор	Описание
`||`	Оператор отношения "ИЛИ". Возвращает `true`, если один из операндов равен `true`. В противном случае возвращает `false`
`&&`	Оператор отношения "И". Возвращает `true`, если оба операнда равны `true`. В противном случае возвращает `false`

Условные операторы

В JScript поддерживается условный оператор `if ... else`. Общий вид этого оператора показан ниже:

```
if (условие)
  выражение_1
[else
  выражение_2]
```

При выполнении оператора `if ... else` оценивается логическое условие, заданное в круглых скобках после ключевого слова `if`. Если в результате оценки условия получилось значение `true`, то выполняется первое выражение. В противном случае выполняется второе выражение (если оно присутствует).

Оператор `if ... else` может быть вложенным. Заметим, что если в первом или втором выражении нужно расположить несколько операторов, то их следует выделить фигурными скобками:

```
if (x == 5) {
  if (y == 6)
    z = 17;
  }
else
  z = 20;
```

В JScript также существует специальный тип условного оператора, который называется оператором "`?:`". В общем виде он записывается так:

```
условие ? выражение_1 : выражение_2
```

При вычислении оператора "`?:`" вначале оценивается условие, расположенное в левой части. Если оно равно `true`, то выполняется первое выражение, в противном случае — второе. Например:

```
hours += (theHour >= 12) ? " PM" : " AM";
```

Операторы циклов

Microsoft JScript поддерживает несколько типов циклов: цикл `for`, цикл `for in`, цикл `while`, цикл `do while`. Рассмотрим каждый из них подробнее.

Цикл *for*

В общем случае оператор цикла `for` имеет три раздела (инициализация, условие и итерация) и записывается следующим образом:

```
for ([инициализация,] [условие,] [итерация]) {
  тело цикла
}
```

В разделе инициализации обычно выполняется присваивание начальных значений переменным цикла. Здесь можно объявлять новые переменные с помощью ключевого слова `var`.

Во втором разделе задается условие выхода из цикла. Это условие оценивается каждый раз при прохождении цикла. Если в результате такой оценки получается логическое значение `true`, то начинают выполняться строки из тела цикла, в противном случае происходит выход из цикла. В том случае, когда условие было ложным с самого начала (при первой проверке), цикл не будет выполнен ни разу.

Раздел итерации применяется для изменения значений переменных цикла (например, увеличения или уменьшения значения счетчика цикла).

Пример использования цикла `for` приведен в листинге 5.1. Отметим, что символы `//`, используемые в этом примере, в языке JScript означают комментарий, т. е. следующие после них символы программой восприниматься не будут.

Листинг 5.1. Пример использования цикла `for`

```
var howFar = 11, // Верхний предел для счетчика цикла
var sum = new Array(howFar), // Массив из 11 элементов, индексы от 0 до 10
var theSum = 0;
sum[0] = 0;
// Цикл выполнится 10 раз
for(var icount = 1; icount < howFar; icount++) {
theSum += icount;
sum[icount] = theSum;
}
var newSum = 0;
// Цикл не выполнится ни разу
for(var icount = 1, icount > howFar; icount++) {
newSum += icount,
}
```

```
var sum = 0,
// Бесконечный цикл
for(var icount = 1; icount > 0; icount++) {
sum += icount;
}
```

Цикл *for ... in*

Оператор цикла `for ... in` предназначен для просмотра всех свойств объекта Для каждого свойства указанный цикл выполняет операторы, содержащиеся в теле цикла

```
for (переменная in объект) {
  тело цикла
}
```

Цикл `for . in` можно использовать для вывода на экран всех свойств объекта в одном цикле

```
function objectDisplay(obj) {
  var displayLine;
  for (var prop in obj) {
    displayLine=obj name+"."+prop+"="+obj[prop];
    WScript.Echo(displayLine)
  }
 WScript Echo ("------------");
}
```

Для вывода строки на экран здесь используется метод `Echo` объекта `WScript`, который будет описан ниже в *разд "Стандартные объекты WSH"* этой главы

Цикл *while*

Цикл `while` похож на цикл `for` В нем также условие выхода из цикла проверяется перед выполнением итерации, однако в цикле `while`, в отличие от `for`, нет встроенного счетчика и выражения, его изменяющего

Оператор `while` записывается в следующем виде

```
while (условие) {
  Тело цикла
}
```

Пример использования цикла `while` приведен в листинге 5 2

Листинг 5.2. Пример использования цикла `while`

```
Var theMoments = "",
Var theCount = 42; // Начальное значение счетчика цикла
```

```
while (theCount >= 1) {
  if (theCount > 1) {
    theMoments = "До взрыва осталось " + theCount + " сек!";
    }
  else {
    theMoments = "Осталась секунда!";
  }
  theCount--; // Изменение значения счетчика цикла.
}
theMoments = "ВЗРЫВ!";
```

Цикл *do ... while*

Этот цикл является примером цикла с постусловием и записывается в следующем виде

```
do {
  тело цикла
}
while (условие);
```

В этом случае цикл выполняется до тех пор, пока проверяемое после ключевого слова `while` условие не станет ложным (`false`) Так как условие проверяется уже после прохождения тела цикла, то операторы внутри цикла `do ... while` выполнятся по крайней мере один раз

Пример использования цикла `do .. while` приведен в листинге 5 3.

Листинг 5.3. Пример использования цикла `do ... while`

```
var howFar = 11; // Верхний предел для счетчика цикла
var sum = new Array(howFar); // Массив из 11 элементов, индексы от 0 до 10
var theSum = 0;
sum[0] = 0;
var icount = 1;
// Цикл выполнится 10 раз
do {
theSum += icount;
sum[icount] = theSum;
icount++;
}
while (icount < howFar);
```

Внутри цикла любого вида можно применять два специальных оператора `break` и `continue`

Оператор *break*

С помощью оператора `break` можно прервать выполнение цикла в любом месте; управление при этом передастся на оператор, следующий сразу за циклом

```
var i = 0;
while (i < 100) {
  if (i == 50)
     break;
     i++;
}
i++; // Значение i станет равным 51
```

Оператор *continue*

Оператор `continue` прерывает текущую итерацию цикла и начинает новую. В различных видах циклов этот оператор производит следующие действия

- в циклах `while` и `do...while` проверяется условие цикла и если оно равно `true`, то вновь выполняется тело цикла;
- в цикле `for` изменяется значение счетчика в разделе итерации, проверяется условие цикла и если оно равно `true`, то тело цикла выполняется вновь,
- в цикле `for...in` переменная цикла переходит к следующему полю объекта, и тело цикла выполняется вновь

Пример использования оператора `continue`

```
var s = "", i=0;
while (i < 10) {
  i++; // Skip 5
  if (i==5) {
    continue;
  }
  s += i;
}
```

Прочие операторы

Рассмотрим еще несколько часто применяемых операторов (табл 5 10)

Таблица 5.10. *Прочие операторы*

Оператор	Описание
	Точка Применяется для доступа к свойству объекта или для вызова его метода

Таблица 5.10 (окончание)

Оператор	Описание
[]	Квадратные скобки Применяются для индексирования массива
()	Скобки Применяются либо для изменения порядка вычисления выражений, либо для передачи параметров функциям
,	Запятая Применяется для многократных вычислений

С помощью оператора "," можно, например, в разделе итерации цикла `for` изменять значение сразу нескольких переменных:

```
var i, j;
j = 10;
for (i = 0; i<=10; i++, j--) {
       . . .
}
```

Порядок выполнения операторов

В табл. 5.11 операторы языка JScript расположены по старшинству, т. е. в составных операторах первыми будут выполняться те из них, которые стоят в этой таблице выше. Если операторы расположены в одной строке таблицы, то они выполняются слева направо.

Таблица 5.11. Порядок выполнения операторов

Оператор	Описание
. [] ()	Доступ к полю объекта, индексирование в массиве, вызов функции
++ -- - ~ !	Унарные операторы
* / %	Умножение, деление, вычисление остатка от деления
+ - +	Сложение, вычитание, конкатенация строк
<< >> >>>	Битовые сдвиги
< <= > >=	Меньше, меньше или равно, больше, больше или равно
== !=	Равенство, неравенство
&	Логическое И
^	Логическое ИСКЛЮЧАЮЩЕЕ ИЛИ
|	Логическое ИЛИ
&&	Оператор отношения "И"

Таблица 5.11 (окончание)

Оператор	Описание
`||`	Оператор отношения "ИЛИ"
`?:`	Условный оператор
`= += -= *= .= %= >>= >>>= <<= |= &= ^=`	Присваивание
`,`	Многократное вычисление

Функции

Функции в JScript, как и в других алгоритмических языках, позволяют объединить несколько операций под одним именем. В случае необходимости функция может быть вызвана из любого места сценария.

В сценариях JScript поддерживаются два вида функций: встроенные функции и функции пользователя, которые вы пишете сами.

Встроенные функции

В табл. 5.12 описаны встроенные функции языка JScript, которые в сценарии можно вызывать в любом месте без предварительного описания.

Таблица 5.12. Встроенные функции

Функция	Описание
`escape(`*`charstring`*`)`	Кодирование строки *charstring* с применением URL-кодировки. При этом все специальные неотображаемые символы пребразуются к виду %xx, где xx — шестнадцатеричный код символа
`eval(`*`codestring`*`)`	Интерпретация и выполнение кода JScript, содержащегося в строке *codestring*. Эта функция позволяет динамически создавать текст сценария
`isFinite(`*`number`*`)`	Возвращает `true`, если параметр *number* является корректным числом. В противном случае возвращает `false`
`isNaN(`*`numvalue`*`)`	Возвращает `true`, если параметр *numvalue* имеет специальное значение NaN (*см. описание числового типа*). В противном случае возвращает `false`. Эту функцию можно применять для оценки значений, возвращаемых функциями преобразования типов `parseInt` и `parseFloat`

Таблица 5.12 (окончание)

Функция	Описание
`parseFloat(numstring)`	Преобразовывает строку *numstring* в число с плавающей точкой
`parseInt(numstring,[radix])`	Преобразовывает строку *numstring* в целое число. Целочисленный параметр *radix* может принимать значения от 2 до 36, которые указывают основание счисления для числа, содержащегося в *numstring*. Если *radix* не указан, то строки с префиксом '0x' задают шестнадцатеричные числа, а строки с префиксом '0' — восьмеричные. Все остальные строки определяют десятичные числа.
`unescape(charstring)`	Выполняет действие, противоположное действию функции escape, т. е. перекодирует строку *charstring* из URL-кодировки в текстовую строку

Функции пользователя

Функции, определяемые пользователем, могут находиться в любом месте сценария и иметь произвольное число параметров (аргументов). Общий вид определения функции имеет вид:

```
function Имя_функции([параметр1] [, параметр2] [..., параметрN]) {
  ...
  Тело функции
  ...
  [return значение;]
}
```

Ключевое слово `return` позволяет функции вернуть значение любого допустимого типа. Например, приведенная ниже функция `MyFunction` возвращает `true`, если оба ее аргумента меньше 10:

```
function MyFunction(x,y) {
  if ((x<10) && (y<10))
    return true
  else
     return false;
}
```

Встроенные объекты (классы)

Как уже отмечалось в самом начале раздела, JScript является объектно-ориентированным языком, поэтому работа со строками, датой и временем, а также такими структурами, как массивы и коллекции, осуществляется с по-

мощью соответствующих встроенных объектов. Кроме этого, математические вычисления выполняются также с помощью специального встроенного объекта. В табл. 5.13 описаны некоторые объекты, которые могут быть полезны при создании сценариев с помощью WSH.

Таблица 5.13. *Некоторые встроенные объекты JScript*

Объект	Описание
`Array`	Создание и работа с массивами данных произвольного типа
`Date`	Работа с данными, содержащими дату или время
`Enumerator`	Работа с коллекциями данных произвольного типа
`Math`	Выполнение математических вычислений
`String`	Работа с текстовыми строками

Для того чтобы в сценарии использовать встроенный объект, необходимо создать переменную, с помощью которой можно будет получить доступ к свойствам и методам этого объекта. Для создания большинства переменных такого вида применяется оператор `new` и специальная функция — конструктор нужного объекта. Название конструктора всегда совпадает с названием соответствующего встроенного объекта. Приведем пример создания объектов `Date` и `Array`:

```
var d;
d = new Date();
var a;
a = new Array(10);
```

Отметим, что объекты `String` можно создавать, просто записывая в кавычках значение строки:

```
var s;
s = "Привет!";
```

Опишем объекты, приведенные в табл. 5.13 более подробно.

Объект *Array*

Новый объект встроенного класса `Array` можно создать с помощью оператора `new` следующими способами:

`new Array()` — создание массива нулевой длины

`new Array(N)` — создание массива длины `N`

`new Array(a0, a1, ..., aN)` — создание массива длины `N+1` с элементами *a0*, *a1*, ..., *aN*

Например

```
var A1, A2, A3;
A1 = new Array();
A2= new Array(3);
A3 = new Array(0,"Строка",2 5);
```

Нумерация элементов в массивах всегда начинается с нуля После того как массив создан и проинициализирован, обращаться к его элементам можно с помощью обычного оператора индексации `[]`, например

```
A3[1] = A3[0] + A3[2];
```

Длину массива, т е число содержащихся в нем элементов, можно узнать с помощью свойства `length` объекта `Array` Для того чтобы динамически изменить длину массива (уменьшить или увеличить), достаточно просто записать соответствующее значение в свойство `length`

```
var A,
A = new Array(1,2,3,4,5); // Длина массива A равна 5
A.length = 3, // Теперь длина массива A равна 3
```

Некоторые наиболее часто используемые методы встроенного объекта `Array` описаны в табл 5 14

Таблица 5.14. *Методы объекта* `Array`

Метод	Описание
`a1.concat(a2)`	Возвращает новый массив, являющийся результатом объединения (склеивания) двух массивов *a1* (его элементы идут первыми) и *a2* (его элементы идут после элементов массива *a1*)
`join(separator)`	Возвращает строку, содержащую все идущие друг за другом элементы массива, разделенные символом, указанным в параметре *separator*
`reverse()`	Располагает элементы массива в обратном порядке (первый меняется местами с последним, второй — с предпоследним и т д) Новый массив при этом не создается
`slice(start,[end])`	Возвращает часть массива, начиная с элемента с индексом *start* и заканчивая элементом с индексом *end* Если в качестве *end* указано отрицательное число, то оно задает смещение от конца массива Если параметр *end* не указан, то берутся все элементы массива, начиная с элемента с индексом *start*

Таблица 5.14 (окончание)

Метод	Описание
`sort([sortfunction])`	Возвращает массив с отсортированными элементами Параметр *sortfunction* определяет имя функции, используемой для сортировки, если этот параметр опущен, то сортировка производится в порядке увеличения ASCII-кодов элементов массива

Пример использования методов объекта `Array` приведен в листинге 5 4

Листинг 5.4. Пример использования методов объекта `Array`

```
var A1, A2, A3,
A1 = new Array(2),
A2 = new Array(2,3,4,5);
A1[0] = 0;
A1[1] = 1;
A3 = A2.concat(A1); // A3=(2,3,4,5,0,1)
A3 sort(), // A3=(0,1,2,3,4,5)
```

Объект *Date*

Для создания нового объекта встроенного класса `Date` используется один из трех конструкторов

Конструктор первого вида позволяет создать объект, в котором хранится информация о текущей дате и времени·

```
var d;
d = new Date();
```

Здесь время задается по Гринвичу, т е с использованием времени Universal Coordinated Time (UCT)

Конструктор второго вида имеет единственный параметр·

```
var d;
d = new Date(nMilliseconds);
```

Параметр `nMilliseconds` задает дату в миллисекундах, считая от 1 января 1970 года

Конструктор третьего вида предназначен для раздельного задания компонентов даты и имеет следующий вид

```
var d;
d = new Date(year, month, date [, hours [, min [, sec [, ms]]]]);
```

Значения параметров последнего конструктора приведены в табл 5 15.

Таблица 5.15. *Параметры конструктора* `Date`

Параметр	Описание
`year`	Год в четырехзначном формате, например 1998 (но не 98)
`month`	Номер месяца от 0 (январь) до 11 (декабрь)
`date`	Календарная дата в диапазоне от 1 до 31
`hours`	Час дня в диапазоне от 0 до 23
`min`	Минуты в диапазоне от 0 до 59
`sec`	Секунды в диапазоне от 0 до 59
`ms`	Миллисекунды в диапазоне от 0 до 999

Наиболее часто используемые методы объекта `Date` описаны в табл. 5 16

Таблица 5.16. *Некоторые методы объекта* `Date`

Метод	Описание
`getDate()`	Возвращает календарную дату в диапазоне от 1 до 31
`getDay()`	Возвращает номер дня недели (0 для воскресенья, 1 — для понедельника и т д)
`getFullYear()`	Возвращает четырехзначный номер года
`getHours()`	Возвращает число часов (отсчет идет с полуночи)
`getMilliseconds()`	Возвращает число миллисекунд
`getMinutes()`	Возвращает число минут (отсчет идет с начала часа)
`getMonth()`	Возвращает число месяцев (отсчет идет с января)
`getSeconds()`	Возвращает число секунд (отсчет идет с начала минуты)
`getTime()`	Определение времени для объекта `Date` Возвращает количество миллисекунд, прошедших с 1 января 1970 года
`getTimezoneOffset()`	Возвращает смещение локального времени относительно времени по Гринвичу (в миллисекундах)
`parse(dateVal)`	Возвращает число миллисекунд, прошедших с полуночи 1 января 1970 года по время, заданное параметром `dateVal` Для вызова метода `parse` не обязательно создавать объект класса `Date`, достаточно просто сослаться на имя этого класса `n = Date.parse("10 May 2001 13:00:00"),` Параметр `dateVal` может задаваться в нескольких форматах *(подробнее см документацию по языку JScript)*

Таблица 5.16 (продолжение)

Метод	Описание
`setDate(date)`	Устанавливает календарную дату Параметр *date* может принимать любые положительные или отрицательные значения Если значение *date* больше, чем количество дней в месяце, который хранится в объекте `Date`, или *date* является отрицательным числом, то календарная дата устанавливается в число, равное разности параметра *date* и числа дней в этом месяце
`setFullYear(year)`	Устанавливает номер года, заданный параметром *year*
`setHours(hours)`	Устанавливает количество часов, заданное параметром *hours* Параметр *hours* может принимать любые положительные или отрицательные значения (при необходимости происходит соответствующее изменение даты, записанной в объекте класса `Date`)
`setMilliseconds(ms)`	Устанавливает количество миллисекунд, заданное параметром *ms* Параметр *ms* может принимать любые положительные или отрицательные значения (при необходимости происходит соответствующее изменение даты, записанной в объекте класса `Date`)
`setMinutes(min)`	Устанавливает количество минут, заданное параметром *min* Параметр *min* может принимать любые положительные или отрицательные значения (при необходимости происходит соответствующее изменение даты, записанной в объекте класса `Date`)
`setMonth(mon)`	Устанавливает номер месяца, прошедшего с начала года Параметр *mon* может принимать любые положительные или отрицательные значения (при необходимости происходит соответствующее изменение даты, записанной в объекте класса `Date`)
`setSeconds(sec)`	Устанавливает количество секунд, заданное параметром *sec* Параметр *sec* может принимать любые положительные или отрицательные значения (при необходимости происходит соответствующее изменение даты, записанной в объекте класса `Date`)
`setTime(ms)`	Устанавливает дату, соответствующую количеству миллисекунд (параметр *ms*), прошедших с 1 января 1970 года
`toGMTString()`	Преобразует дату в строку и возвращает результат в стандартном формате времени по Гринвичу (Greenwich Mean Time, GMT)
`ToLocaleString()`	Преобразует дату в строку и возвращает результат в формате локального времени

Таблица 5.16 (окончание)

Метод	Описание
`ToUTCString()`	Преобразует дату в строку и возвращает результат в формате UTC
`UTC (year, month, date[, hours[, min [, sec [,ms]]]])`	Преобразует дату, заданную параметрами метода, в количество миллисекунд, прошедшее с полуночи 1 января 1970 года. При использовании этого метода, как и метода `parse`, объект класса `Date` создавать не обязательно `n = Date.UTC(year,month,date),`

Пример использования методов объекта `Date` приведен в листинге 5.5.

Листинг 5.5. Пример использования методов объекта `Date`

```
var d;
var s = "";
d = new Date();
s = "Дата: " + d.getDate() + "." + d.getMonth() + "." + d.getYear();
s += "\n";
s += "Время: " + d.getHours() + ":" + d.getMinutes() + ":" +
d.getSeconds();
```

После выполнения этих строк в переменной `s` будут записаны текущие дата и время.

Объект *Enumerator*

С помощью объекта `Enumerator` можно получить доступ к любому элементу *коллекции* (в VBScript для этого служит цикл `For ..Each`). Коллекцией в языке JScript называется множество элементов, которое отличается от массива тем, что к элементам коллекции нельзя получить прямой доступ с помощью индексов — можно только перемещать указатель текущего элемента на самый первый или следующий относительно текущего элемент.

Замечание

Для объектов WSH термин "коллекция" будет иметь несколько иной смысл — там с помощью метода `Item` можно будет получить доступ к элементу по его индексу или названию.

Для создания нового объекта встроенного класса `Enumerator` используется конструктор следующего вида:

```
var e;
e = new Enumerator(collection);
```

Здесь параметр collection указывает на коллекцию, для доступа к элементам которой и создается объект класса Enumerator Сами коллекции обычно являются свойствами других объектов

Методы объекта Enumerator представлены в табл 5 17 (свойств у этого объекта нет)

Таблица 5.17. *Методы объекта* `Enumerator`

Метод	Описание
atEnd()	Возвращает true, если указатель текущего элемента находится на элементе, следующем за последним экземпляром коллекции, либо коллекция пуста, либо текущий элемент не определен В противном случае возвращается false
item()	Возвращает значение текущего элемента коллекции Если коллекция пуста или текущий элемент не определен, возвращается неопределенное значение undefined
moveFirst()	Перемещает указатель на первый элемент коллекции Если в коллекции нет элементов, текущий элемент принимает неопределенное значение undefined
moveNext()	Перемещает указатель на следующий элемент коллекции Если перед применением этого метода указатель находился на последнем элементе коллекции, либо коллекция пуста, текущий элемент принимает неопределенное значение undefined

Пример, поясняющий схему применения объекта Enumerator, приведен в листинге 5 6

Листинг 5.6. Пример использования объекта `Enumerator`

```
var fso, s, n, e, x;
// Создание объекта FileSystemObject
fso = WScript.CreateObject("Scripting.FileSystemObject");
// Создание объекта Enumerator для доступа к коллекции fso.Drives
e = new Enumerator(fso.Drives);
s = "";
// Цикл для просмотра всех элементов коллекции
for (;!e.atEnd();e.moveNext()) {
  // Извлечение элемента коллекции
  x = e.item();
  s = s + x.DriveLetter;
  s += " - ";
  if (x.DriveType == 3)
    n = x.ShareName;
```

```
  else
     if (x.IsReady)
       n = x.VolumeName;
     else
       n = "[Устройство не готово]";
  s +=  n + "\n";
}
```

Здесь с помощью метода `CreateObject` объекта `WScript` (этот объект будет описан ниже в *разд. "Стандартные объекты WSH"* данной главы) создается экземпляр объекта `FileSystemObject` (переменная `fso`), с помощью которого можно получить доступ к файловой системе компьютера (объект `FileSystemObject` будет описан в гл. 6). Свойство `Drives` этого объекта является коллекцией, содержащей сведения обо всех доступных устройствах (дисках).

В цикле мы формируем строку `s`, в которой будут записаны буквы всех дисков компьютера и имена этих дисков (в случае локального диска — это метка диска, а в случае сетевого — имя диска в UNC-формате).

Объект *Math*

Встроенный класс `Math` применяется для математических вычислений и содержит основные математические константы и функции.

Замечание

Объект `Math` создается сервером сценариев автоматически и не может быть создан при помощи оператора `new`, как другие встроенные объекты Все методы и свойства этого объекта доступны из сценария без какого-либо предварительного объявления

Свойства объекта `Math` (все они являются математическими константами) описаны в табл. 5.18.

Таблица 5.18. *Свойства объекта* `Math`

Свойство	Описание
`E`	Константа e. Приблизительное ее значение равно 2,72
`LN2`	Натуральный логарифм числа 2 (приблизительно 0,693)
`LN10`	Натуральный логарифм числа 10 (приблизительно 2,302)
`LOG2E`	Логарифм числа e по основанию 2 (примерно 1,442)
`LOG10E`	Логарифм числа e по основанию 10 (примерно 0,434)
`PI`	Число ?, т. е константа с приблизительным значением, равным 3,14

Таблица 5.18 (окончание)

Свойство	Описание
SQRT1_2	Корень квадратный из ½ (примерно 0,707)
SQRT2	Корень квадратный из 2 (примерно 1,414)

Методы объекта Math (они являются стандартными математическими функциями) приведены в табл. 5.19.

Таблица 5.19. Методы объекта Math

Метод	Описание
abs(x)	Возвращает абсолютное значение числа x
acos(x)	Возвращает арккосинус числа x
asin(x)	Возвращает арксинус числа x
atan(x)	Возвращает арктангенс числа x
atan2(y,x)	Вычисляет угол в радианах от оси Ox до точки (y,x). Возвращаемое значение лежит в диапазоне от $-\pi$ до π
ceil(x)	Возвращает наименьшее целое значение, большее или равное аргументу x
cos(x)	Возвращает косинус числа x
exp(x)	Экспоненциальная функция Возвращает число e, возведенное в степень x
floor(x)	Возвращает наибольшее целое значение, меньшее или равное аргументу x
log(x)	Возвращает натуральный логарифм числа x
max(x1,x2)	Возвращает наибольшее из двух значений x1 и x2
min(x1,x2)	Возвращает наименьшее из двух значений x1 и x2
pow(y,x)	Возводит число y в степень x и возвращает полученный результат
random()	Возвращает случайное число в интервале от 0 до 1
round(x)	Выполняет округление значения аргумента x до ближайшего целого Если десятичная часть числа равна 0,5 или больше этого значения, то округление выполняется в бóльшую сторону, иначе — в меньшую
sin(x)	Возвращает синус числа x

Таблица 5.19 (окончание)

Метод	Описание
`sqrt(x)`	Вычисляет квадратный корень из числа `x` и возвращает полученное значение
`tan(x)`	Возвращает тангенс числа `x`

Приведем пример использования свойств и методов встроенного объекта `Math`:

```
var x,y,z;
x = 12;
y = Math.sqrt(x);
z = 2*Math.PI*Math.pow(y,x);
```

Объект *String*

Встроенный объект `String` предназначен для выполнения различных операций над текстовыми строками. Обычно объекты класса `String` создаются просто с помощью записи в переменную текстового литерала:

```
var s1, s2;
s1 = "Это строка";
s2 = "Это тоже строка";
```

Также можно создавать такие объекты с помощью оператора `new`:

```
var s1, s2;
s1 = new String("Это строка");
s2 = new String("Это тоже строка");
```

Объект `String` имеет свойство `length`, в котором хранится длина строки.

Некоторые методы объекта `String`, не связанные с тегами HTML, приведены в табл. 5.20.

Таблица 5.20. Некоторые методы объекта `String`

Метод	Описание
`charAt(`*index*`)`	Возвращает символ с индексом *index* из строки. Нумерация символов в строке начинается с нуля, поэтому допустимыми значениями параметра *index* являются числа из диапазона от нуля до числа, на единицу меньшего длины строки

Таблица 5.20 *(продолжение)*

Метод	Описание
`charCodeAt(index)`	Выбирает символ с индексом *index* из строки и возвращает этот символ в кодировке Unicode
`s1.concat(s2)`	Возвращает строку, являющуюся результатом конкатенации (склеивания) строк `s1` и `s2` (то же самое, что `s1+s2`)
`fromCharCode(c1,...,cN)`	Возвращает строку, состоящую из Unicode-символов с кодами *c1*, ..., *cN*
`s.indexOf(substr, startindex)`	Возвращает индекс символа, с которого начинается первое вхождение подстроки *substr* в строку *s*. Если подстрока не найдена, возвращается –1. Параметр *startindex* задает номер символа, с которого следует начинать поиск. Если этот параметр не задан, то поиск производится с начала строки. Поиск производится слева направо
`s.lastIndexOf(substr, startindex)`	То же самое, что метод `indexOf`, но поиск производится справа налево, т. е. возвращается номер последнего символа, с которого начинается вхождение подстроки *substr* в строку `s`
`s.match(rgExp)`	Возвращает в виде массива результат поиска в строке `s` подстроки, задаваемой регулярным выражением *rgExp* (поиск с использованием регулярных выражений описан в документации по JScript)
`replace(rgExp, replaceText)`	Возвращает копию строки, в которой произведены необходимые замены текста. Шаблон для поиска задается регулярным выражением *rgExp*, строка для замены — параметром *replaceText*. Первоначальная строка методом `replace` не изменяется
`search(rgExp)`	Возвращает номер первого символа в строке, с которого начинается подстрока, удовлетворяющая регулярному выражению *rgExp*
`slice(start, [end])`	Возвращает часть строки, начиная с позиции *start* и заканчивая символом, стоящим в позиции с номером *end* (или последним символом в строке, если параметр `end` опущен). Если в качестве *end* указано отрицательное число, то этот параметр задает смещение от конца массива

Таблица 5.20 (окончание)

Метод	Описание
`s.split(str)`	Возвращает массив строк, полученных в результате разбиения строки `s` на подстроки. Параметр *str* задает строку или объект `Regular Expression`, которые определяют символ, являющийся признаком начала разбиения
`substr(start [, length])`	Возвращает подстроку, начинающуюся с позиции *start* и имеющую длину *length*. Если в качестве *length* указано отрицательное число или ноль, то возвращается пустая строка
`substring(start, end)`	Возвращается подстрока, состоящая из символов, начинающихся с позиции *start* и заканчивающихся позицией *end*. В качестве *start* и *end* могут быть указаны строки, которые в этом случае автоматически преобразуются в числа
`toLowerCase()`	Возвращает строку, в которой все алфавитные символы преобразованы к нижнему регистру
`toUpperCase()`	Возвращает строку, в которой все алфавитные символы преобразованы к верхнему регистру

Приведем пример использования методов объекта `String`:

```
var s1, s2, s3;
s1 = "Первая строка";
s2 = "Вторая строка";
s1 = s1.toUpperCase();
s2 = s2.substr(0,6);
s1 = s1.slice(7);
s3= s2 + " " + s1;
```

После выполнения этих строк значением переменной `s3` будет строка `Вторая строка`.

Стандартные объекты WSH

В этом разделе мы опишем собственную объектную модель WSH. С помощью внутренних объектов WSH из сценариев можно выполнять следующие основные задачи:

- выводить информацию в стандартный выходной поток (на экран) или в окно Windows;

- читать данные из стандартного входного потока (т. е. вводить данные с клавиатуры) или использовать информацию, выводимую другой командой;
- использовать свойства и методы внешних объектов, а также обрабатывать события этих объектов;
- запускать новые процессы или активизировать уже имеющиеся;
- работать с локальной сетью: определять имя зарегистрировавшегося пользователя, подключать сетевые диски и принтеры;
- просматривать и изменять переменные среды;
- получать доступ к специальным папкам Windows;
- создавать ярлыки Windows;
- работать с системным реестром.

В WSH входят следующие объекты:

1. `WScript`. Это главный объект WSH, который служит для создания других объектов или связи с ними, содержит сведения о сервере сценариев, а также позволяет вводить данные с клавиатуры и выводить информацию на экран или в окно Windows.
2. `WshArguments`. Обеспечивает доступ к параметрам командной строки запущенного сценария или ярлыка Windows.
3. `WshEnvironment`. Предназначен для работы с переменными среды.
4. `WshSpecialFolders`. Обеспечивает доступ к специальным папкам Windows.
5. `WshNetwork`. Используется при работе с локальной сетью: содержит сетевую информацию для локального компьютера, позволяет подключать сетевые диски и принтеры.
6. `WshShell`. Позволяет запускать процессы, создавать ярлыки, работать с переменными среды, системным реестром и специальными папками Windows.
7. `WshShortcut`. Позволяет работать с ярлыками Windows.
8. `WshUrlShortcut`. Предназначен для работы с ярлыками сетевых ресурсов.

Кроме этого, имеется объект `FileSystemObject`, обеспечивающий доступ к файловой системе компьютера (этот объект будет описан в гл. 6).

Перейдем теперь к рассмотрению свойств и методов внутренних объектов WSH.

Объект *WScript*

Свойства объекта `WScript` позволяют получить полный путь к используемуся серверу сценариев (`wscript.exe` или `cscript.exe`), параметры команд-

ной строки, с которыми запущен сценарий, режим его работы (интерактивный или пакетный) Кроме этого, с помощью свойств WScript можно выводить информацию в стандартный выходной поток и читать данные из стандартного входного потока Также объект WScript предоставляет методы для работы внутри сценария с объектами автоматизации и вывода информации на экран (в текстовом режиме) или в окно Windows.

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

Свойства объекта WScript представлены в табл 5 21

Таблица 5.21. *Свойства объекта* WScript

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

Опишем более подробно те свойства объекта WScript, которые требуют дополнительных пояснений.

Свойство *Arguments*

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

```
var objArgs = WScript.Arguments;
for (i=0; i<=objArgs.Count()-1; i++)
  WScript.Echo(objArgs(i));
```

Свойства *StdErr, StdIn, StdOut*

Доступ к стандартным входным и выходным потокам с помощью свойств `StdIn`, `StdOut` и `StdErr` можно получить только в том случае, если сценарий запускался в консольном режиме с помощью `cscript.exe`. Если сценарий был запущен с помощью `wscript.exe`, то при попытке обратиться к этим свойствам возникнет ошибка "`Invalid Handle`" (перечень ошибок выполнения сценариев в WSH приведен в *приложении 2*).

Работать с потоками `StdOut` и `StdErr` можно с помощью методов `Write`, `WriteLine`, `WriteBlankLines`, а с потоком `StdIn` — с помощью методов `Read`, `ReadLine`, `ReadAll`, `Skip`, `SkipLine`. Эти методы кратко описаны в табл. 5 22.

***Таблица 5.22.** Методы для работы с потоками*

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

Напомним, что операционная система Windows поддерживает механизм конвейеризации (символ "`|`" в командной строке) Этот механизм делает воз-

можным передачу данных от одной программы к другой. Таким образом, используя стандартные входные и выходные потоки, можно из сценария обрабатывать строки вывода другого приложения или перенаправлять выводимые сценарием данные на вход программ-фильтров (FIND или SORT). Например, следующая команда будет сортировать строки вывода сценария example.js и выводить их в файл sort.txt:

```
cscript //nologo example.js | sort > sort.txt
```

Опция //nologo нужна для того, чтобы в файл sort.txt не попадали строки с информацией о разработчике и номере версии WSH.

Кроме этого, с помощью методов, работающих с входным потоком StdIn, можно организовывать диалог с пользователем, т. е. создавать интерактивные сценарии. Например:

```
var s;
WScript.StdOut.WriteLine("Как Вас зовут?");
s = WScript.StdIn.ReadLine();
WScript.StdOut.WriteLine("Здравствуйте, " + s + "!");
```

Объект WScript имеет несколько методов, которые описаны в табл 5 23

Таблица 5.23. *Методы объекта* WScript

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

Таблица 5.23 (окончание)

Метод	Описание
`Sleep(intTime)`	Приостанавливает выполнения сценария (переводит его в неактивное состояние) на заданное параметром `intTime` число миллисекунд

Приведем дополнительные пояснения и примеры использования для методов, рассмотренных в табл. 5 23.

Метод *CreateObject*

Если указан необязательный параметр `strPrefix`, то после создания объекта в сценарии можно обрабатывать события, возникающие в этом объекте (естественно, если объект предоставляет интерфейсы для связи с этими событиями) Когда объект сообщает о возникновении определенного события, сервер сценариев вызывает функцию, имя которой состоит из префикса `strPrefix` и имени этого события Например, если в качестве `strPrefix` указано `MYOBJ_`, а объект сообщает о возникновении события `OnBegin`, то будет запущена функция `MYOBJ_OnBegin`, которая должна быть описана в сценарии.

В следующем примере метод `CreateObject` используется для создания объекта `WshNetwork`:

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

Отметим, что объекты автоматизации из сценариев JScript можно создавать и без помощи WSH, используя объект `ActiveXObject` языка JScript, например:

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

Метод *ConnectObject*

Объект, соединение с которым осуществляется с помощью метода `ConnectObject`, должен предоставлять интерфейс к своим событиям

В следующем примере в переменной `MyObject` создается абстрактный объект `SomeObject`, затем из сценария вызывается метод `SomeMetod` этого объекта После этого устанавливается связь с переменной `MyObject` и задается префикс `MyEvent` для процедур обработки события этого объекта Если в объекте возникнет событие с именем `Event`, то будет вызвана функция `MyEvent_Event` Метод `DisconnectObject` объекта `WScript` производит отсоединение объекта `MyObject`.

```
var MyObject = WScript.CreateObject("SomeObject");
MyObject.SomeMethod();
WScript.ConnectObject(MyObject,"MyEvent");
```

```
function MyEvent_Event(strName){
  WScript.Echo(strName);
  }

WScript.DisconnectObject(MyObject);
```

Метод *DisconnectObject*

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

Метод *Echo*

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

В следующем примере на выходное устройство будут выведены пустая строка, три числа и строка текста:

```
WScript.Echo();
WScript.Echo(1,2,3);
WScript.Echo("Привет!");
```

Метод *Sleep*

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

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

Объекты-коллекции

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

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

Объект *WshArguments*

Объект WshArguments содержит коллекцию параметров командной строки запущенного сценария или ярлыка Windows. Этот объект можно создать только с помощью свойства Arguments объектов WScript и WshShortcut.

В принципе, работать с элементами коллекции можно стандартным для JScript образом — создать объект Enumerator и использовать его методы moveNext, item и atEnd. Например, вывести на экран все параметры командной строки, с которыми запущен сценарий, можно следующим образом:

```
var objArgs, e, x;
objArgs = WScript.Arguments;
e = new Enumerator(objArgs);
for (;!e.atEnd();e.moveNext()) {
  x = e.item();
  WScript.Echo(x);
  }
```

Однако намного удобнее использовать методы Count и Item самого объекта WshArguments (эти методы имеются у всех коллекций WSH). Метод Count возвращает число элементов в коллекции, т. е. количество аргументов командной строки, а метод Item(n) — значение n-го элемента коллекции (нумерация начинается с нуля). Более того, чтобы получить значение отдельного элемента коллекции WshArguments, можно просто указать его индекс в круглых скобках после имени объекта.

Замечание

Число элементов в коллекции хранится и в свойстве Length объекта WshArguments

Таким образом, предыдущий пример можно переписать более компактным образом:

```
var
i, objArgs=WScript.Arguments;
for (i=0;i<=objArgs.Count()-1;i++)
WScript.Echo(objArgs(i));
```

Объект *WshEnvironment*

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

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

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

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

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

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

Кроме этого, у объекта `WshEnvironment` имеется метод `Remove(strName)`, который удаляет заданную переменную среды. Например, следующий сценарий удаляет две переменные (`EXAMPLE_1` и `EXAMPLE_2`) из окружения среды пользователя:

```
var WshShell = WScript.CreateObject("WScript.Shell");
var WshUsrEnv = WshShell.Environment("User");
WshUsrEnv.Remove("EXAMPLE_1");
WshUsrEnv.Remove("EXAMPLE_2");
```

Объект *WshSpecialFolders*

Объект `WshSpecialFolders` обеспечивает доступ к коллекции, содержащей пути к специальным папкам Windows (например, к рабочему столу или меню **Пуск**); задание путей к таким папкам может быть необходимо, например, для создания ярлыков на рабочем столе непосредственно из сценария. Этот объект создается с помощью свойства `SpecialFolders` объекта `WshShell`:

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

Как и все коллекции WSH, объект `WshSpecialFolders` имеет свойство `Length` и методы `Count` и `Item`. Доступ к отдельному элементу производится либо через имя соответствующей папки, либо через числовой индекс:

```
var WshShell, WshFldrs, i;
WshShell = WScript.CreateObject("Wscript.Shell");
WshFldrs = WshShell.SpecialFolders;
```

```
WScript.Echo("Некоторые специальные папки...");
WScript.Echo("Desktop="+ WshFldrs.Item("Desktop"));
WScript.Echo("Favorites="+ WshFldrs("Favorites"));
WScript.Echo("Programs="+ WshFldrs("Programs"));
WScript.Echo("");
WScript.Echo("Список всех специальных папок...");
for (i=0;i<= WshFldrs.Count()-1;i++){
  WScript.Echo(WshFldrs(i));
}
```

Перечень специальных папок, путь к которым можно получить с помощью объекта `WshSpecialFolders`, будет приведен ниже при описании метода `SpecialFolders` объекта `WshShell`.

Работа с сетью и оболочкой Windows

Для работы с локальной сетью и оболочкой Windows (специальные папки, переменные среды, системный реестр) предназначены соответственно объекты `WshNetwork` и `WshShell`.

Объект *WshNetwork*

Объект `WshNetwork` предназначен для работы с ресурсами локальной сети; с помощью методов этого объекта можно подключать и отключать сетевые диски и принтеры.

Объект `WshNetwork` создается следующим образом:

```
var objNet=WScript.CreateObject( "WScript.Network")
```

Свойства данного объекта приведены в табл. 5 24.

Таблица 5.24. *Свойства объекта* `WshNetwork`

Свойство	Описание
`ComputerName`	Содержит имя компьютера, на котором запущен сценарий
`UserDomain`	Содержит имя домена, в котором зарегистрировался пользователь
`UserName`	Содержит имя пользователя

В следующем примере на экран выводятся сетевое имя компьютера и имя зарегистрировавшегося на нем пользователя:

```
var objNet = WScript.CreateObject("WScript.Network");
WScript.Echo("Имя машины:",objNet.ComputerName);
WScript.Echo("Имя пользователя:",objNet.UserName);
```

Методы объекта WshNetwork описаны в табл 5 25

Таблица 5.25. *Методы объекта* WshNetwork

Метод	Описание
AddPrinterConnection(*strLocalName, strRemoteName [,bUpdateProfile] [,strUser] [,strPassword]*)	Подключает локальный порт компьютера к сетевому принтеру
Для Windows NT/Windows 2000 AddWindowsPrinterConnection(*strPrnPath*) Для Windows 9x AddWindowsPrinterConnection (*strPrnPath, strDriverName[,strPort]*)	Регистрирует принтер в Windows и подключает его к сетевому ресурсу В отличие от AddPrinterConnection, этот метод позволяет создать связь с сетевым принтером без явного перенаправления вывода в локальный порт
EnumNetworkDrives()	Возвращает коллекцию, в которой хранятся буквы и сетевые пути ко всем подключенным сетевым дискам
EnumPrinterConnections()	Возвращает коллекцию, в которой хранятся данные о всех подключенных сетевых принтерах
MapNetworkDrive(*strLocalName, strRemoteName, [bUpdateProfile], [strUser], [strPassword]*)	Подключает сетевой ресурс *strRemoteName* под локальным именем диска *strLocalName*
RemoveNetworkDrive(*strName, [bForce], [bUpdateProfile]*)	Отключает подключенный сетевой диск
RemovePrinterConnection(*strName, [bForce], [bUpdateProfile]*)	Отключает подключенный сетевой принтер
SetDefaultPrinter(*strPrinterName*)	Делает заданный сетевой принтер принтером по умолчанию

Опишем методы из табл 5 25 более подробно.

Метод *AddPrinterConnection*

Если необязательный параметр *bUpdateProfile* равен true, то создаваемое сетевое подключение будет сохранено в профиле пользователя

Параметры *strUser* (имя пользователя) и *strPassword* (пароль) нужны в том случае, когда вы подключаете сетевой принтер от имени пользователя, которое отличается от имени текущего пользователя, зарегистрированного в системе

В следующем примере метод AddPrinterConnection применяется для подключения принтера с сетевым именем \\Server1\Epson к локальному порту LPT1

```
var WshNetwork = CreateObject("WScript.Network");
WshNetwork.AddPrinterConnection("LPT1", "\\Server1\Epson");
```

Метод *AddWindowsPrinterConnection*

Параметр *strDriverName* указывает имя драйвера, необходимого для подключаемого принтера Если принтер подключается в операционной системе Windows 9*x*, то нужный драйвер уже должен быть установлен на этой машине, иначе возникнет ошибка подключения В Windows NT/2000 параметр *strDriverName* игнорируется

Параметр *strPort* задает в явном виде порт, вывод в который будет перенаправлен на сетевой ресурс (по умолчанию это порт LPT1) В Windows NT/2000 параметр *strPort* игнорируется

В следующем примере метод AddWindowsPrinterConnection применяется для подключения сетевого принтера к локальному порту LPT1 (по умолчанию):

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

Метод *EnumNetworkDrives*

Элементами возвращаемой коллекции являются буквы, обозначающие имеющиеся сетевые диски и сетевые имена ресурсов, к которым эти диски подключены Первым элементом коллекции является буква, вторым — сетевое имя; эта последовательность сохраняется для всех сетевых дисков в коллекции

В следующем примере на экран выводятся буквы, обозначающие все сетевые диски и имена ресурсов, к которым они подключены

```
var WshNetwork = WScript.CreateObject("WScript.Network");
var oDrives = WshNetwork.EnumNetworkDrives();
for (i=0; i<=oDrives.Count()-1;i++)
  WScript.Echo(oDrives.Item(i));
```

Метод *EnumPrinterConnections*

Элементами возвращаемой коллекции являются названия локальных портов и сетевые имена принтеров, связанных с этими портами Сама коллекция организована так же, как и коллекция, возвращаемая методом EnumNetworkDrives

В следующем примере на экран выводятся названия всех переназначенных портов и имена сетевых ресурсов, с которыми они связаны:

```
var WshNetwork = WScript.CreateObject("WScript.Network");
var oPrinters = WshNetwork.EnumPrinterConnections();
for (i=0; i<=oPrinters.Count()-1;i++)
  WScript.Echo(oPrinters.Item(i));
```

Метод *MapNetworkDrive*

Если необязательный параметр *bUpdateProfile* равен `true`, то создаваемое сетевое подключение будет сохранено в профиле пользователя.

Параметры *strUser* (имя пользователя) и *strPassword* (пароль) нужны в том случае, когда вы подключаете сетевой диск от имени пользователя, которое отличается от имени текущего пользователя, зарегистрированного в системе.

В следующем примере диск `Z` подключается к сетевому ресурсу `\\Server1\Programs`:

```
var WshNetwork = WScript.CreateObject("WScript.Network");
WshNetwork.MapNetworkDrive("Z:","\\Server1\Programs");
```

Метод *RemoveNetworkDrive*

В качестве параметра *strName* может быть указано либо локальное имя (буква сетевого диска), либо сетевое имя (имя подключенного сетевого ресурса); это зависит от того, каким образом осуществлялось подключение. Если сетевому ресурсу сопоставлена буква локального диска, то параметр *strName* должен быть локальным именем. Если сетевому ресурсу не сопоставлена никакая буква, то параметр *strName* должен быть сетевым именем

Если необязательный параметр *bForce* равен `true`, то отключение сетевого ресурса будет произведено вне зависимости от того, используется этот ресурс в настоящее время или нет.

Если необязательный параметр *bUpdateProfile* равен `true`, то отключаемое сетевое подключение будет удалено из профиля пользователя.

В следующем примере производится подключение диска `Z` к сетевому ресурсу, а затем отключение этого ресурса:

```
var WshNetwork = WScript.CreateObject("WScript.Network");
WshNetwork.MapNetworkDrive("Z:","\\Server1\Programs");
WshNetwork.RemoveNetworkDrive("Z:");
```

Метод *RemovePrinterConnection*

В качестве параметра *strName* может быть указано либо локальное имя (название порта), либо сетевое имя (имя подключенного сетевого принтера); это зависит от того, каким образом осуществлялось подключение. Если се-

тевому ресурсу явным образом сопоставлен локальный порт (например, LPT1), то параметр *strName* должен быть локальным именем. Если сетевому принтеру не сопоставлен локальный порт, то параметр *strName* должен быть сетевым именем.

Параметры *bForce* и *bUpdateProfile* в этом методе имеют то же значение, что и одноименные параметры в методе RemoveNetworkDrive

В следующем примере отключается сетевой принтер, который был назначен на порт LPT1:

```
var WshNetwork = WScript.CreateObject("WScript.Network");
WshNetwork.RemovePrinterConnection("LPT1:");
```

Метод *SetDefaultPrinter*

Параметр *strName* задает сетевое имя принтера, который должен будет использоваться в системе по умолчанию.

В следующем примере с помощью метода AddPrinterConnection к порту LPT1 подключается сетевой принтер \\Server1\Epson, который затем устанавливается принтером по умолчанию:

```
var WshNetwork = WScript.CreateObject("WScript.Network");
WshNetwork.AddPrinterConnection("LPT1:","\\Server1\Epson");
WshNetwork.SetDefaultPrinter("\\Server1\Epson");
```

Объект *WshShell*

С помощью объекта WshShell можно запускать новый процесс, создавать ярлыки, работать с системным реестром, получать доступ к переменным среды и специальным папкам Windows Создается этот объект следующим образом:

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

Объект WshShell имеет два свойства, приведенные в табл. 5.26.

Таблица 5.26. *Свойства объекта* WshShell

Свойство	Описание
Environment	Содержит объект WshEnvironment, который обеспечивает доступ к переменным среды операционной системы для Windows NT/2000 или к переменным среды текущего командного окна для Windows *9x*

Таблица 5.26 *(окончание)*

Свойство	Описание
SpecialFolders	Содержит объект WshSpecialFolders для доступа к специальным папкам Windows (рабочий стол, меню **Пуск** и т. д.)

Опишем теперь методы, имеющиеся у объекта WshShell (табл. 5.27).

Таблица 5.27. *Методы объекта* WshShell

Метод	Описание
AppActivate(*title*)	Активизирует заданное параметром *title* окно приложения. Строка *title* задает название окна (например, "calc" или "notepad")
CreateShortcut(*strPathname*)	Создает объект WshShortcut для связи с ярлыком Windows (расширение lnk) или объект WshUrlShortcut для связи с сетевым ярлыком (расширение url) Параметр *strPathname* задает полный путь к создаваемому или изменяемому ярлыку
Environment(*strType*)	Возвращает объект WshEnvironment, содержащий переменные среды заданного вида
ExpandEnvironmentStrings(*strString*)	Возвращает значение переменной среды текущего командного окна заданной строкой *strString* (имя переменной должно быть окружено знаками "%")
LogEvent(*intType, strMessage [,strTarget]*)	Протоколирует события в журнале Windows NT или в файле WSH.log. Целочисленный параметр *intType* определяет тип сообщения, строка *strMessage* — текст сообщения. Параметр *strTarget* может задаваться только в Windows NT, он определяет название системы, в которой протоколируются события (по умолчанию это локальная система). Метод LogEvent возвращает true, если событие записано успешно, и false — в противном случае

Таблица 5.27 *(окончание)*

Метод	Описание
`Popup(strText,[nSecToWait], [strTitle], [nType])`	Выводит на экран информационное окно с сообщением, заданным параметром *strText*. Параметр *nSecToWait* задает количество секунд, по истечении которых окно будет автоматически закрыто, параметр *strTitle* определяет заголовок окна, параметр *nType* указывает тип кнопок и иконки для окна
`RegDelete(strName)`	Удаляет из системного реестра заданную переменную или ключ целиком
`RegRead(strName)`	Возвращает значение ключа или переменной реестра
`RegWrite(strName, anyValue [,strType])`	Записывает в реестр значение заданной переменной или ключа целиком
`Run(strCommand, [intWindowStyle], [bWaitOnReturn])`	Создает новый процесс, который запускает приложение, заданное параметром *strCommand*
`SendKeys(string)`	Посылает одно или несколько нажатий клавиш в активное окно (эффект тот же, как если бы вы нажимали эти клавиши на клавиатуре)
`SpecialFolders(strSpecFolder)`	Возвращает строку, содержащую путь к специальной папке Windows, заданной параметром *strSpecFolder*

Рассмотрим методы, приведенные в табл. 5.27 более подробно.

Метод *AppActivate*

Метод `AppActivate` активизирует уже запущенное указанное приложение (устанавливает на него фокус), но не производит никаких действий по изменению размеров его окна. Для того чтобы первоначально запустить нужное приложение и определить вид его окна, надо использовать метод `Run` объекта `WshShell`. Для того чтобы определить, какое именно приложение необходимо активизировать, строка *title* сравнивается по очереди с названиями окон всех запущенных приложений. Если не найдено ни одного точного совпадения, будет производиться поиск того приложения, название окна которого начинается со строки `title`. Если и в этом случае не будет найдено ни одного подходящего приложения, то будет вестись поиск при-

ложения, заголовок которого заканчивается на эту строку Если будет найдено несколько подходящих окон, то произойдет активизация одного из них (окно выбирается произвольно)

В качестве примера использования метода AppActivate в листинге 5 7 приведен сценарий runcalc.js, который запускает стандартный калькулятор Windows и выполняет в нем несколько простых арифметических действий (для этого используется метод SendKeys)

Листинг 5.7. Сценарий runcalc.js

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

Метод *CreateShortcut*

Этот метод позволяет создать новый или открыть уже существующий ярлык для изменения его свойств

В листинге 5.8 приведен пример сценария, в котором создаются два ярлыка — на сам выполняемый сценарий (объект oShellLink) и на сетевой ресурс (oUrlLink)

Листинг 5.8. Пример создания ярлыков из сценария

```
var WshShell=WScript.CreateObject("WScript Shell"),
    oShellLink=WshShell.CreateShortcut("Current Script.lnk");
oShellLink.TargetPath=WScript.ScriptFullName,
oShellLink.Save();

var oUrlLink = WshShell.CreateShortcut("Microsoft Web Site.URL");
oUrlLink.TargetPath = "http://www.microsoft.com";
oUrlLink.Save(),
```

Метод *Environment*

Параметр *strType* задает вид переменных среды, которые будут записаны в коллекции WshEnvironment, возможными значениями этого параметра яв-

ляются System (переменные среды операционной системы), User (переменные среды пользователя), Volatile (временные переменные) или Process (переменные среды текущего командного окна)

Замечание

Для Windows 9x единственным допустимым значением параметра *strType* является "Process"

В следующем примере мы распечатываем число процессоров, имеющихся в компьютере с операционной системой Windows NT (переменная NUMBER_OF_PROCESSORS), и путь к каталогу Windows:

```
var WshShell = WScript.CreateObject("WScript.Shell"),
    WshSysEnv = WshShell.Environment("SYSTEM");
WScript.Echo(WshSysEnv("NUMBER_OF_PROCESSORS"));
WScript.Echo(WshShell.Environment.Item("WINDIR"));
```

Метод *ExpandEnvironmentString*

В следующем примере на экран выводится путь к каталогу Windows:

```
var WS = WScript.CreateObject("WScript.Shell");
WScript.Echo("Каталог Windows:"+WS.ExpandEnvironmentStrings("%WinDir%"));
```

Метод *LogEvent*

В Windows NT события записываются в системном журнале, а в Windows 9x — в файле WSH.log, расположенном в каталоге пользователей Windows. Запись в WSH.log будет содержать время события, его тип и текст. Типы сообщений описаны в табл. 5.28.

Таблица 5.28. *Типы сообщений (параметр intType)*

Код	Значение	Код	Значение
0	SUCCESS	4	INFORMATION
1	ERROR	8	AUDIT_SUCCESS
2	WARNING	16	AUDIT_FAILURE

В следующем примере производится протоколирование работы сценария регистрации (здесь предполагается, что если этот сценарий отработал успешно, то функция RunLoginScript возвращает true, в противном случае — false)

```
var WshShell = WScript.CreateObject("WScript.Shell");
rc = RunLoginScript();
```

```
if (rc)
  WshShell.LogEvent(0,"Logon Script Completed Successfully");
else
  WshShell.LogEvent(1,"Logon Script failed");
```

Метод *Popup*

Если в методе не задан параметр *strTitle*, то заголовком окна будет "Windows Script Host."

Параметр *nType* может принимать те же значения, что и в функции MessageBox из Microsoft Win32 API. В табл. 5.29 описаны некоторые возможные значения параметра *nType* и их смысл (полный список значений этого параметра можно посмотреть в описании функции MessageBox в документации по функциям Windows API).

Таблица 5.29. *Типы кнопок и иконок для метода* Popup

Значение *nType*	Описание
0	Выводится кнопка **Ok**
1	Выводятся кнопки **Ok** и **Cancel**
2	Выводятся кнопки **Abort**, **Retry** и **Ignore**
3	Выводятся кнопки **Yes**, **No** и **Cancel**
4	Выводятся кнопки **Yes** и **No**
5	Выводятся кнопки **Retry** и **Cancel**
16	Выводится иконка **Stop Mark**
32	Выводится иконка **Question Mark**
48	Выводится иконка **Exclamation Mark**
64	Выводится иконка **Information Mark**

Естественно, в методе Popup можно комбинировать значения параметра, приведенные в табл. 5.29. Например, в результате выполнения следующего сценария:

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

на экран будет выведено информационное окно, показнное на рис. 5.3, которое автоматически закроется через 5 секунд.

Метод Popup возвращает целое значение, с помощью которого можно узнать, какая именно кнопка была нажата для выхода (табл. 5.30).

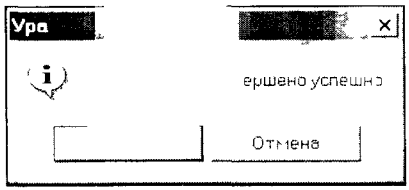


Рис. 5.3. *Информационное окно, созданное методом* `Popup`

Таблица 5.30. *Возвращаемые методом* `Popup` *значения*

Значение	Описание
–1	Пользователь не нажал ни на одну из кнопок в течение времени, заданного параметром `nSecToWait`
1	Нажата кнопка **Ok**
2	Нажата кнопка **Cancel**
3	Нажата кнопка **Abort**
4	Нажата кнопка **Retry**
5	Нажата кнопка **Ignore**
6	Нажата кнопка **Yes**
7	Нажата кнопка **No**

Метод *RegDelete*

Если параметр `strName` оканчивается символами "\\", то этот метод удаляет ключ целиком (вместе со всеми переменными внутри его), в противном случае удаляется только одна заданная переменная. Параметр `strName` должен начинаться с одного из следующих корневых ключей (табл. 5.31).

Таблица 5.31. *Названия ключей*

Краткое название	Длинное название
HKCU	HKEY_CURRENT_USER
HKLM	HKEY_LOCAL_MACHINE
HKCR	HKEY_CLASSES_ROOT
	HKEY_USERS
	HKEY_CURRENT_CONFIG

В следующем примере из реестра удаляются переменная HKCU\ScriptEngine\Value и ключ HKCU\ScriptEngine\Key:

```
var WshShell = WScript.CreateObject("WScript.Shell");
WshShell.RegDelete("HKCU\\ScriptEngine\\Value");
WshShell.RegDelete("HKCU\\ScriptEngine\\Key\\");
```

Метод *RegRead*

С помощью этого метода можно прочитать типы данных, такие как: REG_SZ, REG_EXPAND_SZ, REG_DWORD, REG_BINARY и REG_MULTI_SZ. Если в реестре содержатся данные других типов, то метод RegRead вернет значение DISP_E_TYPEMISMATCH.

В следующем примере на экран выводятся считанные из реестра с помощью метода RegRead значение переменной и значение ключа:

```
var WS = WScript.CreateObject("WScript.Shell");
WScript.Echo(WS.RegRead("HKCU\\Control Panel\\Keyboard\\KeyboardSpeed"));
WScript.Echo(WS.RegRead("HKCU\\Control Panel\\Keyboard\\"));
```

Метод *RegWrite*

Параметр *anyValue* метода задает значение, которое нужно записать в заданный параметром *strName* ключ или переменную реестра. Необязательный параметр *strType* определяет тип записываемого значения. В качестве *strType* можно указывать REG_SZ, REG_EXPAND_SZ, REG_DWORD и REG_BINARY. Если в качестве параметра *strType* передается другое значение, то метод RegWrite вернет E_INVALIDARG.

В случае, когда *strType* установлено в REG_SZ или REG_EXPAND_SZ, метод RegWrite автоматически конвертирует параметр *anyValue* в строку. Если значение *strType* равно REG_DWORD, то *anyValue* переводится в целый формат. Если *strType* равно REG_BINARY, то *anyValue* должно быть целым числом.

Метод *Run*

Параметр *intWindowStyle* устанавливает вид окна для запускаемого приложения (табл. 5.32).

Таблица 5.32. *Типы окна (intWindowStyle)*

Параметр	Описание
0	Прячет текущее окно и активизирует другое окно
1	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

Таблица 5.32 (окончание)

Параметр	Описание
2	Активизирует окно и отображает его в минимизированном (свернутом) виде
3	Активизирует окно и отображает его в максимизированном (развернутом) виде
4	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	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 Этот флаг должен указываться сценарием, если производится восстановление свернутого окна
10	Устанавливает режим отображения, опирающийся на режим программы, которая запускает приложение

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 же *bWaitOnReturn* установлен в `true`, то сценарий возобновит работу только после завершения вызванного процесса.

При этом если параметр *bWaitOnReturn* равен `true`, то метод `Run` возвращает код выхода вызванного приложения. Если же *bWaitOnReturn* равен `false` или не задан, то метод `Run` всегда возвращает ноль.

В следующем примере мы запускаем Microsoft Notepad и открываем в нем файл с выполняемым сценарием:

```
var WshShell = WScript.CreateObject("WScript.Shell");
WshShell.Run("%windir%\\notepad " + WScript.ScriptFullName);
```

Следующий сценарий печатает код выхода вызванного приложения:

```
var WshShell = WScript.CreateObject("WScript.Shell");
Return = WshShell.Run("notepad " + WScript.ScriptFullName, 1, true);
WScript.Echo("Код возврата:", Return);
```

Метод *SendKeys*

Каждая клавиша задается одним или несколькими символами. Например, для того чтобы задать нажатие друг за другом букв А, Б и В, нужно указать в качестве параметра для SendKes строку "АБВ": *string*="АБВ".

Несколько символов 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. 5.33.

Таблица 5.33. *Коды специальных клавиш для* SendKeys

Клавиша	Код	Клавиша	Код
<Backspace>	{BACKSPACE}, {BS} или {BKSP}	<Стрелка вниз>	{DOWN}
<Break>	{BREAK}	<Стрелка вправо>	{RIGHT}
<Caps Lock>	{CAPSLOCK}	<F1>	{F1}
<Del> или <Delete>	{DELETE} или {DEL}	<F2>	{F2}
<End>	{END}	<F3>	{F3}
<Enter>	{ENTER} или ~	<F4>	{F4}
<Esc>	{ESC}	<F5>	{F5}
<Help>	{HELP}	<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}
<Стрелка вверх>	{UP}	<F15>	{F15}
<Стрелка влево>	{LEFT}	<F16>	{F16}

Для задания комбинаций клавиш с <Shift>, <Ctrl> или <Alt> перед соответствующей клавишей нужно поставить один или несколько кодов из табл. 5.34.

Таблица 5.34. *Коды клавиш <Shift>, <Ctrl> и <Alt>*

Клавиша	Код
<Shift>	+
<Ctrl>	^
<Alt>	%

Для того чтобы задать комбинацию клавиш, которую нужно набирать, удерживая нажатыми клавиши <Shift>, <Ctrl> или <Alt>, необходимо 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 же, чтобы задать одновременное нажатие клавиш <Shift> и <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, `{LEFT 42}` означает нажатие клавиши <Стрелка влево> 42 раза подряд; `{h 10}` означает нажатие клавиши <h> 10 раз подряд.

Замечание

Метод `SendKeys` не может быть использован для посылки нажатий клавиш для приложений, которые не были разработаны специально для запуска в Microsoft Windows (например, для приложений MS-DOS).

Пример, иллюстрирующий использование `SendKeys`, был приведен выше при описании метода `AppActivate`.

Метод *SpecialFolders*

В Windows 9*x* поддерживаются следующие имена специальных папок: `Desktop`, `Favorites`, `Fonts`, `MyDocuments`, `NetHood`, `PrintHood`, `Programs`, `Recent`, `SendTo`, `StartMenu`, `Startup`, `Templates`. В Windows NT/2000 дополнительно можно получить доступ еще к четырем папкам: `AllUsersDesktop`, `AllUsersStartMenu`, `AllUsersPrograms`, `AllUsersStartup`. В следующем примере на экран выводится название папки рабочего стола:

```
var WshShell = WScript.CreateObject("WScript.Shell");
WScript.Echo(WshShell.SpecialFolders("Desktop"));
```

Работа с ярлыками

Свойства и методы для работы с ярлыками Windows предоставляют два объекта WSH: `WshShortcut` и `WshUrlShortcut`.

Объект *WshShortcut*

С помощью объекта `WshShortcut` можно создать новый ярлык Windows или изменить свойства уже существующего ярлыка Данный объект можно создать только посредством метода `CreateShortcut` объекта `WshShell`. В следующем примере из сценария создается ярлык на этот самый сценарий (ярлык будет находиться в текущем каталоге):

```
var WshShell = WScript.CreateObject("WScript.Shell");
var oShellLink = WshShell.CreateShortcut("Current Script.lnk");
oShellLink.TargetPath = WScript.ScriptFullName;
oShellLink.Save();
```

Свойства объекта `WshShortcut` описаны в табл. 5.35.

Таблица 5.35. *Свойства объекта* `WshShortcut`

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

Приведем необходимые пояснения и примеры использования свойств объекта `WshShortcut`.

Свойство *Arguments*

В следующем примере создается ярлык для текущего сценария с двумя параметрами:

```
var WshShell = WScript.CreateObject("WScript.Shell");
var oShellLink = WshShell.CreateShortcut("Current Script.lnk");
oShellLink.TargetPath = WScript.ScriptFullName;
oShellLink.Arguments = "-a abc.txt";
oShellLink.Save();
```

Свойство *HotKey*

Для того чтобы назначить ярлыку горячую клавишу, необходимо в свойство `HotKey` записать строку, содержащую названия нужных клавиш, разделенные символом "+".

Замечание

Горячие клавиши могут быть назначены только ярлыкам, которые расположены на рабочем столе Windows или в меню **Пуск**

В следующем примере на рабочем столе создается ярлык для Блокнота, которому назначается горячая клавиша (комбинация <Ctrl>+<Alt>+<D>):

```
var WshShell = WScript.CreateObject("WScript.Shell");
strDesktop = WshShell.SpecialFolders("Desktop");
oMyShortcut = WshShell.CreateShortcut(strDesktop+"\\a_key.lnk");
oMyShortcut.TargetPath = "%windir%\\notepad.exe";
oMyShortcut.Hotkey = "CTRL+ALT+D";
oMyShortcut.Save();
WScript.Echo(oMyShortcut.Hotkey);
```

Свойство *IconLocation*

Для того чтобы задать иконку для ярлыка, необходимо в свойство `IconLocation` записать строку следующего формата: *путь, индекс*. Здесь параметр *путь* определяет расположение файла, содержащего нужную иконку, а параметр *индекс* — номер этой иконки в файле (номера начинаются с нуля).

В следующем примере создается ярлык на выполняющийся сценарий с первой иконкой (индекс 0) из файла `notepad.exe`:

```
var WshShell = WScript.CreateObject("WScript.Shell");
var oShellLink = WshShell.CreateShortcut("Current Script.lnk");
oShellLink.TargetPath = WScript.ScriptFullName;
oShellLink.IconLocation = "notepad.exe, 0";
oShellLink.Save();
```

Свойство *WindowStyle*

Значением свойства `WindowStyle` является целое число `intWindowStyle`, которое может принимать значения, приведенные в табл. 5.36.

Таблица 5.36. *Значения параметра* `intWindowsStyle`

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

Свойство *WorkingDirectory*

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

```
var WshShell = WScript.CreateObject("WScript.Shell");
var oShellLink = WshShell.CreateShortcut("Notepad.lnk");
oShellLink.TargetPath = "notepad.exe";
oShellLink.WorkingDirectory = "c:\\";
oShellLink.Save();
```

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

Объект *WshUrlShortcut*

С помощью объекта `WshUrlShortcut` можно создать новый ярлык для сетевых ресурсов или изменить свойства уже существующего ярлыка. Этот объект, как и `WshShortcut`, можно создать только посредством метода `CreateShortcut` объекта `WshShell`.

В следующем примере создается сетевой ярлык для сайта **www.microsoft.com**

```
var WshShell = WScript.CreateObject("WScript.Shell");
var oUrlLink = WshShell.CreateShortcut("Microsoft Web Site.URL");
oUrlLink.TargetPath = "http://www.microsoft.com";
oUrlLink.Save();
```

Объект `WshUrlShortcut` имеет два свойства: `FullName` и `TargetPath`, которые полностью аналогичны одноименным свойствам рассмотренного выше объекта `WshShortcut`.

Также у объекта `WshUrlShortcut` имеется метод `Save`, с помощью которого ярлык сохраняется в каталоге, указанном в свойстве `FullName`.

Заключение

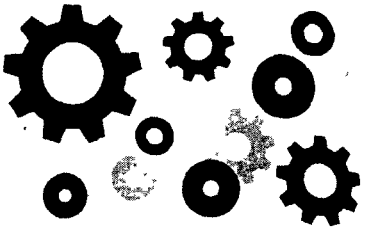
Сервер сценариев WSH позволяет выполнять непосредственно в операционной системе написанные на специализированных языках типа JScript и VBScript сценарии, в которых можно использовать любые зарегистрированные в системе объекты ActiveX. Собственные объекты WSH 2.0 содержат свойства и методы, с помощью которых можно выполнять следующие основные действия:

- выводить информацию в стандартный выходной поток или в окно Windows;
- читать данные из стандартного входного потока, т. е. вводить данные с клавиатуры или использовать информацию, выводимую другой командой;
- использовать свойства и методы внешних объектов, а также обрабатывать события этих объектов;
- запускать новые процессы или активизировать уже имеющиеся;
- работать с локальной сетью: определять имя зарегистрировавшегося пользователя, подключать сетевые диски и принтеры;
- просматривать и изменять переменные среды;
- получать доступ к специальным папкам Windows;
- создавать ярлыки Windows;
- работать с системным реестром.

В следующей главе будут описаны технологии, с помощью которых из сценариев можно работать с файловой системой компьютера, различными базами данных и объектами локальной сети. Также будут рассмотрены сценарии WSH с разметкой XML, которая обеспечивает более мощную среду для написания сценариев.

Зак. 1235

ГЛАВА 6



Практическое использование WSH

Одной из самых распространенных задач, требующих автоматизации, является манипулирование файлами и каталогами. С помощью WSH мы имеем возможность из сценариев получить полную информацию об объектах файловой системы и выполнить необходимые действия над ними. Методы и свойства объектов ActiveX, необходимых для доступа к файловой системе, будут подробно описаны ниже в *разд. "Работа с файловой системой"* данной главы.

Разд. "Доступ к базам данных из сценариев WSH" и *"Сценарии и объекты службы каталогов"* этой главы посвящены вопросам использования в сценариях WSH следующих современных технологий Microsoft:

- ADO (ActiveX Data Objects), с помощью которой можно работать с различными базами данных;
- ADSI (Active Directory Service Interface), которая позволяет осуществить доступ к объектам компьютерных сетей разных производителей.

Здесь не ставится задача более или менее полного раскрытия этих технологий, а лишь очень кратко описываются их основные возможности и приводятся примеры сценариев для их реализации.

Далее в *разд. "Сценарии WSH как приложения XML" этой главы* описывается появившаяся в WSH 2.0 возможность рассматривать сценарии WSH как файлы формата XML (Extensible Markup Language), что делает процесс создания и использования сценария более гибким и удобным.

Работа с файловой системой

Для работы с файловой системой из сценариев предназначены восемь объектов (все они регистрируются при установке WSH), главным из которых является `FileSystemObject`.

С помощью методов объекта FileSystemObject можно выполнять следующие основные действия:

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

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

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

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

В табл. 6.1 кратко описано, какие именно объекты, свойства и методы могут понадобиться для выполнения наиболее часто используемых файловых операций.

Таблица 6.1. *Выполнение основных файловых операций*

Операция	Используемые объекты, свойства и методы
Получение сведений об определенном диске (тип файловой системы, метка тома, общий объем и количество свободного места и т. д.)	Свойства объекта Drive. Сам объект Drive создается с помощью метода GetDrive объекта FileSystemObject
Получение сведений о заданном каталоге или файле (дата создания или последнего доступа, размер, атрибуты и т. д.)	Свойства объектов Folder и File. Сами эти объекты создаются с помощью методов GetFolder и GetFile объекта FileSystemObject
Проверка существования определенного диска, каталога или файла	Методы DriveExists, FolderExists и FileExists объекта FileSystemObject

Таблица 6.1 (окончание)

Операция	Используемые объекты, свойства и методы
Копирование файлов и каталогов	Методы `CopyFile` и `CopyFolder` объекта `FileSystemObject`, а также методы `File.Copy` и `Folder.Copy`
Перемещение файлов и каталогов	Методы `MoveFile` и `MoveFolder` объекта `FileSystemObject`, или методы `File.Move` и `Folder.Move`
Удаление файлов и каталогов	Методы `DeleteFile` и `DeleteFolder` объекта `FileSystemObject`, или методы `File.Delete` и `Folder.Delete`
Создание каталога	Методы `FileSystemObject.CreateFolder` или `Folders.Add`
Создание текстового файла	Методы `FileSystemObject.CreateTextFile` или `Folder.CreateTextFile`
Получение списка всех доступных дисков	Коллекция `Drives`, содержащаяся в свойстве `FileSystemObject.Drives`
Получение списка всех подкаталогов заданного каталога	Коллекция `Folders`, содержащаяся в свойстве `Folder.SubFolders`
Получение списка всех файлов заданного каталога	Коллекция `Files`, содержащаяся в свойстве `Folder.Files`
Открытие текстового файла для чтения, записи или добавления	Методы `FileSystemObject.CreateTextFile` или `File.OpenAsTextStream`
Чтение информации из заданного текстового файла или запись ее в него	Методы объекта `TextStream`

Перейдем теперь к подробному рассмотрению объектов, используемых при работе с файловой системой

Объект *FileSystemObject*

Объект `FileSystemObject` является основным объектом, обеспечивающим доступ к файловой системе компьютера, его методы используются для создания остальных объектов (`Drives`, `Drive`, `Folders`, `Folder`, `Files`, `File` и `TextStream`)

Для создания внутри сценария экземпляра объекта FileSystemObject можно воспользоваться методом CreateObject объекта WScript:

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

Также можно использовать объект ActiveXObject языка JScript (с помощью этого объекта можно работать с файловой системой из сценариев, находящихся внутри HTML-страниц):

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

Объект FileSystemObject имеет единственное свойство Drives, в котором хранится коллекция, содержащая объекты Drive для всех доступных дисков компьютера Примеры, иллюстрирующие использование свойства Drives приведены ниже в *разд "Коллекция Drives"* этой главы

Методы объекта FileSystemObject представлены в табл 6 2

Таблица 6.2. *Методы объекта* FileSystemObject

Метод	Описание
BuildPath(*path, name*)	Добавляет к заданному пути (параметр *path*) новое имя (параметр *name*)
CopyFile(*source, destination* [*, overwrite*])	Копирует один или несколько файлов из одного места (параметр *source*) в другое (параметр *destination*)
CopyFolder(*source,destination* [*,overwrite*])	Копирует каталог со всеми подкаталогами из одного места (параметр *source*) в другое (параметр *destination*)
CreateFolder(*foldername*)	Создает новый каталог с именем *foldername* Если каталог *foldername* уже существует, то произойдет ошибка
CreateTextFile(*filename* [*,overwrite*[*,unicode*]])	Создает новый текстовый файл с именем *filername* и возвращает указывающий на этот файл объект TextStream
DeleteFile(*filespec*[*, force*])	Удаляет файл, путь к которому задан параметром *filespec*
DeleteFolder(*folderspec*[*, force*])	Удаляет каталог, путь к которому задан параметром *folderspec*, вместе со всем его содержимым
DriveExists(*drivespec*)	Возвращает true, если заданное параметром *drivespec* устройство существует, и false — в противном случае
FileExists(*filespec*)	Возвращает true, если заданный параметром *filespec* файл существует, и false — в противном случае

Таблица 6.2 (продолжение)

Метод	Описание
FolderExists(*folderspec*)	Возвращает true, если заданный параметром *folderspec* каталог существует и false — в противном случае
GetAbsolutePathName(*parthspec*)	Возвращает полный путь для заданного относительного пути *pathspec* (из текущего каталога)
GetBaseName(*path*)	Возвращает базовое имя (без расширения) для последнего компонента в пути *path*
GetDrive(*drivespec*)	Возвращает объект Drive, соответствующий диску, заданному параметром *drivespec*
GetDriveName(*path*)	Возвращает строку, содержащую имя диска в заданном пути Если из параметра *path* нельзя выделить имя диска, то метод возвращает пустую строку ("")
GetExtensionName(*path*)	Возвращает строку, содержащую расширение для последнего компонента в пути *patn*. Если из параметра *path* нельзя выделить компоненты пути, то GetExtensionName возвращает пустую строку ("") Для сетевых дисков корневой каталог (\) рассматривается как компонент пути
GetFile(*filespec*)	Возвращает объект File, соответствующий файлу, заданному параметром *filespec*. Если файл, путь к которому задан параметром *filespec*, не существует, то при выполнении метода GetFile возникнет ошибка
GetFileName(*pathspec*)	Возвращает имя файла, заданного полным путем к нему Если из параметра *pathspec* нельзя выделить имя файла, метод GetFileName возвращает пустую строку ("")
GetFolder(*folderpec*)	Возвращает объект Folder, соответствующий каталогу, заданному параметром *folderspec*. Если каталог, путь к которому задан параметром *folderspec*, не существует, при выполнении метода GetFolder возникнет ошибка

Таблица 6.2 (окончание)

Метод	Описание
GetParentFolderName(*path*)	Возвращает строку, содержащую имя родительского каталога для последнего компонента в заданном пути Если для последнего компонента в пути, заданном параметром *path*, нельзя определить родительский каталог, то метод возвращает пустую строку ("")
GetSpecialFolder(*folderpec*)	Возвращает объект Folder для некоторых специальных папок Windows, заданных числовым параметром *folderspec*
GetTempName()	Возвращает случайным образом сгенерированное имя файла или каталога, которое может быть использовано для операций, требующих наличия временного файла или каталога
MoveFile(*source, destination*)	Перемещает один или несколько файлов из одного места (параметр *source*) в другое (параметр *destination*)
MoveFolder(*source, destination*)	Перемещает один или несколько каталогов из одного места (параметр *source*) в другое (параметр *destination*)
OpenTextFile(*filename* [, *iomode* [, *create*[, *format*]]])	Открывает заданный текстовый файл и возвращает объект TextStream для работы с этим файлом

Сами названия методов объекта FileSystemObject довольно прозрачно указывают на выполняемые ими действия Приведем необходимые пояснения и примеры для перечисленных методов

Методы *CopyFile* и *CopyFolder*

Для копирования нескольких файлов или каталогов в последнем компоненте параметра *source* можно указывать групповые символы (? и *), в параметре *destination* групповые символы недопустимы Например, следующий пример является корректным кодом:

```
fso = WScript.CreateObject("Scripting.FileSystemObject");
fso.CopyFile("c:\\mydocuments\\letters\\*.doc", "c:\\tempfolder\\");
```

А так писать нельзя:

```
fso = WScript.CreateObject("Scripting.FileSystemObject");
fso.CopyFile("c:\\mydocuments\\*\\R1???97.xls", "c:\\tempfolder");
```

Необязательный параметр *overwrite* является логической переменной, определяющей, следует ли заменять уже существующий файл/каталог с именем *destination* (*overwrite*=true) или нет (*overwrite*=false)

При использовании методов CopyFile и CopyFolder процесс копирования прерывается после первой возникшей ошибки (как и в команде COPY операционной системы)

Метод *CreateTextFile*

Параметр *overwrite*, используемый в методе, имеет значение в том случае, когда создаваемый файл уже существует Если *overwrite* равно true, то такой файл перепишется (старое содержимое будет утеряно), если же в качестве *overwrite* указано false, то файл переписываться не будет Если этот параметр вообще не указан, то существующий файл также не будет переписан

Параметр *unicode* является логическим значением, указывающим, в каком формате (ASCII или Unicode) следует создавать файл Если *unicode* равно true, то файл создается в формате Unicode, если же *unicode* равно false или этот параметр вообще не указан, то файл создается в режиме ASCII

Для дальнейшей работы с созданным файлом, т е для записи или чтения информации, нужно использовать методы объекта TextStream Например

```
var fso = WScript.CreateObject("Scripting FileSystemObject");
// Создание файла
var a = fso CreateTextFile("c:\\testfile.txt", true);
// Запись строки в файл
a.WriteLine("Привет!");
// Закрытие файла
a Close(),
```

Методы *DeleteFile* и *DeleteFolder*

Параметры *filespec* или *folderspec*, используемые в методах, могут содержать групповые символы ? и * в последнем компоненте пути для удаления сразу нескольких файлов/каталогов

Если параметр *force* равен false или не указан вовсе, то с помощью методов DeleteFile или DeleteFolder будет нельзя удалить файл/каталог с атрибутом "Только для чтения" (Read-Only) Установка для *force* значения true позволит сразу удалять такие файлы/каталоги

Замечание

При использовании метода DeleteFolder неважно, является ли удаляемый каталог пустым или нет — он удалится в любом случае

Если заданный для удаления файл/каталог не будет найден, то возникнет ошибка

Метод *DriveExists*

Для дисководов со съемными носителями метод DriveExists вернет true даже в том случае, если носитель физически отсутствует Для того чтобы определить готовность дисковода, нужно использовать свойство IsReady соответствующего объекта Drive

В качестве примера использования метода DriveExists приведем функцию ReportDriveStatus, которая возвращает информацию о наличии диска, передаваемого в эту функцию в качестве параметра (листинг 6 1)

Листинг 6.1. Функция ReportDriveStatus

```
function ReportDriveStatus(drv)
{
  var fso, s = "";
  fso = WScript.CreateObject("Scripting.FileSystemObject");
  if (fso.DriveExists(drv))
    s += "Диск " + drv + " существует.";
  else
    s += "Диск " + drv + " не существует.";
  return(s);
}
```

Функция ReportDriveStatus будет возвращать информацию о наличии диска, передаваемого в эту функцию в качестве параметра.

Метод *GetAbsolutePathName*

Для иллюстрации работы этого метода предположим, что текущим каталогом является c:\mydocuments\reports В табл 6 3 приведены значения, возвращаемые методом GetAbsolutePathName, при различных значениях параметра *pathspec*

Таблица 6.3. *Варианты работы метода* GetAbsolutePathName

Параметр *pathspec*	Возвращаемое значение
"c:"	"c·\mydocuments\reports"
"c:.."	"c.\mydocuments"
"c \\"	"c:\"

Таблица 6.3 (окончание)

Параметр *pathspec*	Возвращаемое значение
"region1"	"c:\mydocuments\reports\region1"
"c:\\..\\..\\mydocuments"	"c:\mydocuments"

Метод *GetBaseName*

Для иллюстрации работы этого метода приведем следующий пример:

```
var fso, BaseName;
  fso = WScript.CreateObject("Scripting.FileSystemObject");
  BaseName = fso.GetBaseName("C:\\Мои документы\\letter.txt");
  WScript.Echo(BaseName);
```

В результате выполнения описанного сценария на экране будет напечатано слово `letter`.

Метод *GetDrive*

Параметр *drivespec* в данном методе может задаваться одной буквой (например, `C`), буквой с двоеточием (`C:`), буквой с двоеточием и символом разделителя пути (`C:\\`). Для сетевого диска *drivespec* можно указывать в формате UNC (например, `Server1\\Programs`).

Если параметр *drivespec* указывает на несуществующий диск или задан в неверном формате, то при выполнении метода `GetDrive` возникнет ошибка.

Если вам требуется преобразовать строку, содержащую обычный путь к файлу или каталогу, в вид, пригодный для `GetDrive`, необходимо применять методы `GetAbsolutePathName` и `GetDriveName`:

```
DriveSpec = GetDriveName(GetAbsolutePathName(Path))
```

Метод *GetParentFolderName*

Для иллюстрации работы этого метода приведем следующий пример:

```
var fso, ParentFolder;
  fso = WScript.CreateObject("Scripting.FileSystemObject");
  ParentFolder = fso.GetParentFolderName("C:\\Programs\\letter.txt");
  WScript.Echo(ParentFolder);
```

Таким образом, при выполнении сценария на экран выведется строка `C:\Programs`.

Метод *GetSpecialFolder*

Параметр *folderspec* в этом методе является числом и может принимать значения, описанные в табл. 6.4.

Таблица 6.4. *Значения параметра* `folderspec`

Константа	Значение	Описание
`WindowsFolder`	0	Каталог Windows (например, `C:\Windows`)
`SystemFolder`	1	Системный каталог, содержащий файлы библиотек, шрифтов и драйверы устройств
`TemporaryFolder`	2	Каталог для временных файлов, путь к которому хранится в переменной среды `TMP`

Метод *GetTempName*

Метод `GetTempName` только возвращает имя файла, но не создает его. Для создания файла можно воспользоваться методом `CreateTextFile`, подставив в качестве параметра этого метода сгенерированное случайное имя.

Методы *MoveFile* и *MoveFolder*

Как и при использовании методов `CopyFile` и `CopyFolder`, для перемещения нескольких файлов или каталогов в последнем компоненте параметра *source* можно указывать групповые символы (`?` и `*`); в параметре *destination* групповые символы недопустимы.

При использовании методов `MoveFile` и `MoveFolder` процесс перемещения прерывается после первой возникшей ошибки (как и в команде `MOVE` операционной системы). Перемещать файлы и каталоги с одного диска на другой нельзя.

Метод *OpenTextFile*

Числовой параметр *iomode* задает режим ввода/вывода для открываемого файла и может принимать следующие значения (табл. 6.5).

Таблица 6.5. *Параметр* `iomode`

Константа	Значение	Описание
`ForReading`	1	Файл открывается только для чтения, записывать информацию в него нельзя

Таблица 6.5 (окончание)

Константа	Значение	Описание
ForWriting	2	Файл открывается для записи Если файл с таким именем уже существовал, то при новой записи его содержимое теряется
ForAppending	8	Файл открывается для добавления Если файл уже существовал, то информация будет дописываться в конец этого файла

Параметр *create* имеет значение в том случае, когда открываемый файл физически не существует Если *create* равно true, то этот файл создастся, если же в качестве значения *create* указано false или параметр *create* опущен, то файл создаваться не будет

Числовой параметр *format* определяет формат открываемого файла (табл 6 6)

Таблица 6.6. Параметр format

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

Для дальнейшей работы с открытым файлом, т е для записи или чтения информации, нужно использовать методы объекта TextStream

В следующем примере с помощью метода OpenTextFile текстовый файл открывается в режиме добавления информации (листинг 6 2)

Листинг 6.2. Добавление информации в текстовый файл

```
var fs, a, ForAppending;
ForAppending = 8;
fs = new ActiveXObject("Scripting FileSystemObject");
// Открытие файла
a = fs.OpenTextFile("c:\\testfile.txt", ForAppending, false);
// Добавление строки в файл
a.WriteLine("Привет!");
a Close(); // Закрытие файла
```

Замечание

Мнемонические константы, используемые в качестве параметров `iomode` и `create`, можно не определять явным образом в сценарии, как это сделано в вышеприведенном примере, а брать из самого объекта `FileSystemObject` (точнее говоря, из библиотеки типов этого объекта) Для этого в сценариях нужно применять разметку XML, что подробно описано ниже в *разд "Сценарии WSH как приложения XML"* данной главы (*см описание элемента <reference>*)

Объект *Drive*

С помощью объекта `Drive` можно получить доступ к свойствам заданного локального или сетевого диска Создается объект `Drive` с помощью метода `GetDrive` объекта `FileSystemObject` следующим образом

```
var fso, d;
fso = WScript CreateObject("Scripting.FileSystemObject");
d = fso GetDrive("C:"),
```

Также объекты `Drive` могут быть получены как элементы коллекции `Drives`

Свойства объекта `Drive` представлены в табл 6 7, методов у этого объекта нет

Таблица 6.7. *Свойства объекта* `Drive`

Свойство	Описание
`AvailableSpace`	Содержит количество доступного для пользователя места (в байтах) на диске
`DriveLetter`	Содержит букву, ассоциированную с локальным устройством или сетевым ресурсом Это свойство доступно только для чтения
`DriveType`	Содержит числовое значение, определяющее тип устройства 0 — неизвестное устройство, 1 — устройство со сменным носителем, 2 — жесткий диск, 3 — сетевой диск, 4 — CD-ROM, 5 — RAM-диск
`FileSystem`	Содержит тип файловой системы, использующейся на диске (FAT, NTFS или CDFS)
`FreeSpace`	Содержит количество свободного места (в байтах) на локальном диске или сетевом ресурсе Доступно только для чтения

Таблица 6.7 (окончание)

Свойство	Описание
IsReady	Содержит true, если устройство готово, и false — в противном случае Для устройств со сменными носителями и приводов CD-ROM, IsReady возвращает true только в том случае, когда в дисковод вставлен соответствующий носитель и устройство готово предоставить доступ к этому носителю
Path	Содержит путь к диску (например, C , но не C·\)
RootFolder	Содержит объект Folder, соответствующий корневому каталогу на диске Доступно только для чтения
SerialNumber	Содержит десятичный серийный номер тома заданного диска
ShareName	Содержит сетевое имя для диска Если объект не является сетевым диском, то в свойстве ShareName содержится пустая строка (" ")
TotalSize	Содержит общий объем в байтах локального диска или сетевого ресурса
VolumeName	Содержит метку тома для диска Доступно для чтения и записи

В листинге 6 3 приведен сценарий listinfo.js, в котором объект Drive используется для доступа к некоторым свойствам диска C:

Листинг 6.3. Сценарий listinfo.js

```
var fso, d, s;
fso = WScript.CreateObject("Scripting.FileSystemObject"),
d = fso.GetDrive("C:");
WScript.Echo("Информация о диске C:");
WScript.Echo("Серийный номер.",d.SerialNumber),
WScript.Echo("Метка тома·",d.VolumeName);
WScript.Echo("Объем:",d.TotalSize/1024,"kb"),
WScript.Echo("Свободно:",d FreeSpace/1024,"kb");
```

Коллекция *Drives*

Доступная только для чтения коллекция Drives содержит объекты Drive для всех доступных дисков компьютера, в том числе для сетевых дисков и дисководов со сменными носителями

В свойстве Count коллекции Drives хранится число ее элементов, т е число доступных дисков

С помощью метода Item(*drivespec*) можно получить доступ к объекту Drive для диска, заданного параметром *drivespec* Например

```
var fso, DriveCol, d;
fso = WScript.CreateObject("Scripting.FileSystemObject");
DriveCol = fso.Drives;
// Извлечение элемента коллекции
d = DriveCol.Item("C:");
WScript.Echo("На диске C  свободно",d.FreeSpace,"kb"),
```

Для перебора всех элементов коллекции Drives нужно использовать объект Enumerator, который был описан в *разд "Необходимые сведения о языке JScript" гл 5*

В листинге 6 4 приведен файл disksinfo.js, в котором с помощью объекта Enumerator на экран выводятся сведения обо всех доступных дисках

Листинг 6.4. Файл disksinfo.js

```
var fso, s, n, e, x;
fso = WScript.CreateObject("Scripting.FileSystemObject");
// Создание объекта Enumerator для доступа к коллекции Drives
e = new Enumerator(fso Drives);
s = "";
// Цикл для просмотра всех элементов коллекции
for (;!e.atEnd();e.moveNext()) {
  // Извлечение элемента коллекции
  x = e.item();
  s = s + x.DriveLetter;
  s += " - ";
  if (x.DriveType == 3)
    n = x.ShareName;
  else
     if (x.IsReady)
       n = x.VolumeName + "  (" + x.FreeSpace/1024+ ")";
     else
       n = "Устройство не готово";
  s +=  n + "\n";
}
WScript.Echo(s);
```

Объект *Folder*

Объект Folder обеспечивает доступ к свойствам каталога Создать этот объект можно с помощью свойства RootFolder объекта Drive или методов

GetFolder, GetParentFolder и GetSpecialFolder объекта FileSystemObject следующим образом:

```
var fso, folder;
fso = WScript.CreateObject("Scripting.FileSystemObject");
folder = fso.GetFolder("C:\\Мои документы");
```

Также объекты Folder могут быть получены как элементы коллекции Folders.

Свойства объекта Folder представлены в табл. 6.8

Таблица 6.8. *Свойства объекта* `Folder`

Свойство	Описание
Attributes	Позволяет просмотреть или установить атрибуты каталога
DateCreated	Содержит дату и время создания каталога Доступно только для чтения
DateLastAccessed	Содержит дату и время последнего доступа к каталогу Доступно только для чтения
DateLastModified	Содержит дату и время последней модификации каталога Доступно только для чтения
Drive	Содержит букву диска для устройства, на котором находится каталог Доступно только для чтения
Files	Содержит коллекцию Files, состоящую из объектов File для всех файлов в каталоге (включая скрытые и системные)
IsRootFolder	Содержит true, если каталог является корневым, и false — в противном случае
Name	Позволяет просмотреть и изменить имя каталога Доступно для чтения и записи
ParentFolder	Содержит объект Folder для родительского каталога. Доступно только для чтения
Path	Содержит путь к каталогу
ShortName	Содержит короткое имя каталога (в формате 8.3)
ShortPath	Содержит путь к каталогу, состоящий из коротких имен каталогов (в формате 8.3)
Size	Содержит размер всех файлов и подкаталогов, входящих в данный каталог, в байтах
SubFolders	Содержит коллекцию Folders, состоящую из всех подкаталогов каталога (включая подкаталоги с атрибутами "Скрытый" и "Системный")
Type	Содержит информацию о типе каталога

Следующий пример показывает, как объект Folder используется для получения даты создания каталога:

```
var fso, f;
fso = WScript.CreateObject ("Scripting.FileSystemObject");
f = fso.GetFolder("C:\\Мои документы");
WScript.Echo("Дата создания каталога Мои документы: "+f.DateCreated);
```

Методы объекта Folder описаны в табл. 6.9.

Таблица 6.9. *Методы объекта* `Folder`

Метод	Описание
`Copy(destination [,overwrite])`	Копирует каталог в другое место
`CreateTextFile(filename[, overwrite[, unicode]])`	Создает новый текстовый файл с именем *filename* и возвращает указывающий на этот файл объект TextStream (этот метод аналогичен рассмотренному выше методу CreateTextFile объекта FileSystemObject)
`Delete([force])`	Удаляет каталог
`Move(destination)`	Перемещает каталог в другое место

Приведем необходимые замечания для методов из табл. 6.9.

Метод *Copy*

Обязательный параметр *destination* определяет каталог, в который будет производиться копирование; групповые символы в имени каталога недопустимы.

Параметр *overwrite* является логической переменной, определяющей, следует ли заменять уже существующий каталог с именем *destination* (*overwrite*=true) или нет (*overwrite*=false).

Замечание

Вместо метода Copy можно использовать метод CopyFolder объекта FileSystemObject

Метод *Delete*

Если параметр *force* равен false или не указан, то с помощью метода Delete будет нельзя удалить каталог с атрибутом "только для чтения" (read-only). Установка для *force* значения true позволит сразу удалять такие каталоги.

При использовании метода Delete неважно, является ли заданный каталог пустым или нет — он удалится в любом случае

Замечание

Вместо метода Delete можно использовать метод DeleteFolder объекта FileSystemObject

Метод *Move*

Обязательный параметр *destination* определяет каталог, в который будет производиться перемещение; групповые символы в имени каталога недопустимы

Замечание

Вместо метода Move можно использовать метод MoveFolder объекта FileSystemObject

Коллекция *Folders*

Коллекция Folders содержит объекты Folder для всех подкаталогов определенного каталога Создается эта коллекция с помощью свойства SubFolders соответствующего объекта Folder Например, в следующем примере переменная fc является коллекцией, содержащей объекты Folder для всех подкаталогов каталога C:\Program Files

```
var fso, f, fc;
fso = WScript.CreateObject("Scripting FileSystemObject");
f = fso.GetFolder("C:\\Program Files");
fc = f.SubFolders;
```

Коллекция Folders (как и Drives) имеет свойство Count и метод Item Кроме этого, у Folders есть метод Add(*folderName*), позволяющий создавать новые подкаталоги Например

```
var fso, f, fc;
fso = WScript.CreateObject("Scripting.FileSystemObject");
f = fso.GetFolder("C:\\Program Files");
fc = f.SubFolders;
// Создание каталога C:\Program Files\Новая папка
fc.Add("Новая папка");
```

Замечание

Напомним, что новый каталог также можно создать с помощью метода CreateFolder объекта FileSystemObject

Для доступа ко всем элементам коллекции, нужно использовать, как обычно, объект Enumerator. Например, в листинге 6 5 приведен сценарий subfold.js, который выводит на экран названия всех подкаталогов каталога C:\Program Files.

Листинг 6.5. Сценарий subfold.js

```
var fso, f, fc, s;
fso = WScript.CreateObject("Scripting.FileSystemObject");
f = fso.GetFolder("C:\\ProgramFiles");
fc = new Enumerator(f.SubFolders);
s = "";
for (; !fc.atEnd(); fc.moveNext()) {
  s += fc.item();
  s += "\n";
}
WScript.Echo(s);
```

Объект *File*

Объект File обеспечивает доступ ко всем свойствам файла Создать этот объект можно с помощью метода GetFile объекта FileSystemObject следующим образом

```
var fso, f;
fso = WScript.CreateObject("Scripting.FileSystemObject");
f = fso GetFile("C:\\Мои документы\\letter.txt");
```

Также объекты File могут быть получены как элементы коллекции Files.

Свойства объекта File описаны в табл 6 10.

***Таблица 6.10.** Свойства объекта File*

Свойство	Описание
Attributes	Позволяет просмотреть или установить атрибуты файлов
DateCreated	Содержит дату и время создания файла Доступно только для чтения
DateLastAccessed	Содержит дату и время последнего доступа к файлу Доступно только для чтения
DateLastModified	Содержит дату и время последней модификации файла Доступно только для чтения
Drive	Содержит букву диска для устройства, на котором находится файл Доступно только для чтения

Таблица 6.10 (окончание)

Свойство	Описание
Name	Позволяет просмотреть и изменить имя файла Доступно для чтения и записи
ParentFolder	Содержит объект Folder для родительского каталога файла Доступно только для чтения
Path	Содержит путь к файлу
ShortName	Содержит короткое имя файла (в формате 8 3)
ShortPath	Содержит путь к файлу, состоящий из коротких имен каталогов (в формате 8 3)
Size	Содержит размер заданного файла в байтах
Type	Возвращает информацию о типе файла Например, для файла с расширением txt возвратится строка Text Document

Методы объекта File представлены в табл 6 11

Таблица 6.11. Методы объекта File

Метод	Описание
Copy(*destination* [,*overwrite*])	Копирует файл в другое место
Delete([*force*])	Удаляет файл
Move(*destination*)	Перемещает файл в другое место
OpenAsTextStream([*iomode*, [*format*]])	Открывает заданный файл и возвращает объект TextStream, который может быть использован для чтения, записи или добавления данных в текстовый файл

Приведем необходимые замечания для методов из табл 6 11

Метод *Copy*

Обязательный параметр *destination* определяет файл, в который будет производиться копирование, групповые символы в имени файла недопустимы

Параметр *overwrite* является логической переменной, определяющей, следует ли заменять уже существующий файл с именем *destination* (*overwrite*=true) или нет (*overwrite*=false).

Следующий пример иллюстрирует использование метода Copy.

```
var fso, f;
fso = WScript.CreateObject("Scripting.FileSystemObject");
f = fso.CreateTextFile("C:\\testfile.txt", true);
f.WriteLine("This is a test.");
f.Close();
f = fso.GetFile("C:\\testfile.txt");
f.Copy("C:\\Windows\\Desktop\\test2.txt");
```

Замечание

Вместо метода Copy можно использовать метод CopyFile объекта FileSystemObject

Метод *Delete*

Если параметр *force* равен false или не указан, то с помощью метода Delete будет нельзя удалить файл с атрибутом "только для чтения" (read-only) Установка для *force* значения true позволит сразу удалять такие файлы

Замечание

Вместо метода Delete можно использовать метод DeleteFile объекта FileSystemObject

Метод *Move*

Обязательный параметр *destination* определяет файл, в который будет производиться перемещение; групповые символы в имени файла недопустимы.

Замечание

Вместо метода Move можно использовать метод MoveFile объекта FileSystemObject

Метод *OpenAsTextStream*

Числовой параметр *iomode* задает режим ввода/вывода для открываемого файла и может принимать те же значения, что и одноименный параметр в методе OpenTextFile объекта FileSystemObject (см табл 6 5)

Числовой параметр *format* определяет формат открываемого файла (ASCII или Unicode) Этот параметр также может принимать те же значения, что и *format* в методе OpenTextFile объекта FileSystemObject (см табл 6 6).

Замечание

Открыть текстовый файл для чтения можно также с помощью метода `OpenTextFile` объекта `FileSystemObject`

В листинге 6 6 приведен сценарии, иллюстрирующий использование метода `OpenAsTextStream` для записи строки в файл и чтения из него

Листинг 6.6. Запись информации в текстовый файл и чтение из него

```
var fso, f, ts, s;
var ForReading = 1, ForWriting = 2, TristateUseDefault = -2;
fso = WScript.CreateObject("Scripting.FileSystemObject");
// Создание файла в текущем каталоге
fso.CreateTextFile("test1.txt");
// Создание объекта File
f = fso.GetFile("test1.txt");
// Создание объекта TextStream
ts = f.OpenAsTextStream(ForWriting, TristateUseDefault);
// Запись строки в файл
ts.Write("Первая строка");
// Закрытие файла
ts.Close();
// Открытие файла для чтения
ts = f.OpenAsTextStream(ForReading, TristateUseDefault);
// Чтение строки
s = ts.ReadLine();
// Закрытие файла
ts.Close();
WScript.Echo(s);
```

Коллекция *Files*

Коллекция `Files` содержит объекты `File` для всех файлов, находящихся внутри определенного каталога Создается эта коллекция с помощью свойства `Files` соответствующего объекта `Folder` Например, в следующем примере переменная `fc` является коллекцией, содержащей объекты `File` для всех файлов в каталоге `C:\Мои документы`

```
var fso, f, fc;
fso = WScript.CreateObject("Scripting.FileSystemObject");
f = fso.GetFolder("C:\\Мои документы");
fc = f.Files;
```

Как и рассмотренные выше коллекции `Drives` и `Folders`, коллекция `Files` имеет свойство `Count` и метод `Item`

Для доступа в цикле ко всем элементам коллекции Files используется объект Enumerator В качестве примера использования этого объекта в листинге 6 7 приведен сценарий listfiles.js, выводящий на экран названия всех файлов, которые содержатся в каталоге C:\Мои документы

Листинг 6.7. Сценарий listfiles.js

```
var fso, f, f1, fc, s;
fso = WScript.CreateObject("Scripting.FileSystemObject");
f = fso.GetFolder("C:\\Мои документы");
fc = new Enumerator(f.files),
s = "";
for (; !fc.atEnd(); fc.moveNext()) {
  s += fc.item();
  s += "\n";
}
WScript.Echo(s);
```

Объект *TextStream*

Объект TextStream обеспечивает последовательный (строка за строкой) доступ к текстовому файлу. Методы этого объекта позволяют выполнять по отношению к файлам такие операции, как чтение и запись

Создать объект TextStream можно с помощью методов

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

В следующем примере переменная a является объектом TextStream и используется для записи строки текста в файл c·\testfile.txt·

```
var fso = WScript.CreateObject("Scripting.FileSystemObject");
var a = fso.CreateTextFile("c:\\testfile.txt", true);
a.WriteLine("This is a test.");
a.Close();
```

Свойства объекта TextStream описаны в табл 6 12

***Таблица 6.12.** Свойства объекта TextStream*

Свойство	Описание
AtEndOfLine	Содержит true, если указатель достиг конца строки в файле, и false — в противном случае Доступно только для чтения

Таблица 6.12 (окончание)

Свойство	Описание
AtEndOfStream	Содержит true, если указатель достиг конца файла, и false — в противном случае Доступно только для чтения
Column	Содержит номер колонки текущего символа в текстовом файле Доступно только для чтения
Line	Содержит номер текущей строки в текстовом файле Доступно только для чтения

Методы объекта TextStream представлены в табл 6 13

Таблица 6.13. Методы объекта TextStream

Метод	Описание
Close()	Закрывает открытый файл
Read(*n*)	Считывает из файла *n* символов и возвращает полученную строку
ReadAll()	Считывает полностью весь файл и возвращает полученную строку
ReadLine()	Возвращает полностью считанную из файла строку
Skip(*n*)	Пропускает при чтении *n* символов
SkipLine()	Пропускает целую строку при чтении
Write(*string*)	Записывает в файл строку *string* (без символа конца строки)
WriteBlankLines(*n*)	Записывает в файл *n* пустых строк (символы перевода строки и возврата каретки)
WriteLine([*string*])	Записывает в файл строку *string* (вместе с символом конца строки) Если параметр *string* опущен то в файл записывается пустая строка

В листинге 6 8 приведен сценарий example.js, иллюстрирующий использование методов объекта TextStream.

Листинг 6.8. Сценарий example.js

```
var fso, a, s, ForReading = 1;
fso = WScript.CreateObject("Scripting.FileSystemObject");
a = fso.CreateTextFile("c:\\testfile.txt", true);
```

```
a.Write("Это ");
a.WriteLine("первая строка");
a.WriteLine("Это вторая строка");
a.Close();
a = fso.OpenTextFile("c:\\testfile.txt", ForReading);
a.SkipLine();
s = a.ReadLine();
WScript.Echo(s);
```

В результате выполнения этого сценария на экран выведется строка `Это вторая строка`

Пример. Перемещение файлов с протоколированием действий

Поставим перед собой следующую задачу Пусть в каталоге `C:\In` находятся различные файлы Требуется выделить из них все файлы с заданным расширением `485` и перенести их в каталог `D·\Out` При этом для каждого переносимого файла нужно записать в файл `C:\Logs\log.txt` следующую информацию· имя файла, дату и время его создания, дату и время перемещения файла Структура файла `log.txt` должна быть следующей

Имя файла (Дата и время создания) Дата и время перемещения

Например

```
34556.485     (19/12/2000 10·45)  19/12/2000 11:02
43432_KL.485  (19/12/2000 10.45)  19/12/2000 11:02
45.485        (19/12/2000 10·45)  19/12/2000 11:02
                ...
```

Кроме этого, во время перемещения файлов на экран должна выводиться информация о том, какой именно файл обрабатывается, а после завершения работы сценария нужно напечатать общее количество перемещенных файлов

Поставленная задача решается путем выполнения сценария `movelog.js`, приведенного в листинге 6 9, для его запуска нужно использовать `cscript.exe`.

Листинг 6.9. Сценарий `movelog.js`

```
var
  Source="C:\\In\\", // Путь к каталогу-источнику файлов для перемещения
  Dest="D:\\Out\\",  // Путь к целевому каталогу
  Mask="485",        // Расширение файлов для перемещения
  PathLog="C:\\Logs\\log.txt"; // Путь к log-файлу
var FSO,Fold,Fils,Fil,enumObj,s,ss,FLog,d,sl;
var ForAppending=8;
```

```
// Создание объекта FileSystemObject
FSO=WScript.CreateObject("Scripting.FileSystemObject");
// Создание объекта Folder для каталога-источника
Fold=FSO.GetFolder(Source);
Fils=Fold.Files; // Создание коллекции файлов каталога-источника
FLog=FSO.OpenTextFile(PathLog,ForAppending,true); // Открытие log-файла
WriteLog(); // Запись информации в log-файл
MoveFiles(); // Перемещение нужных файлов
FLog.Close(); // Закрытие log-файла

/*************************************************/
/* Функция WriteLog - запись информации в log-файл */
/*************************************************/
function WriteLog() {
  // Вывод информации на экран
  WScript.Echo("");
  WScript.Echo("Пишем в log-файл...");
  // Создание объекта Enumerator для коллекции Fils
  enumObj = new Enumerator(Fils);
  while (!enumObj.atEnd()) {  // Цикл для выделения нужных файлов
    Fil=enumObj.item(); // Создание объекта File для элемента коллекции
    s=FSO.GetExtensionName(Source+Fil.name); // Выделение расширения файла
    if (s==Mask) { // Определение нужного файла
      WScript.Echo("  "+Fil.name); // Вывод на экран имени файла
      // Формирование строки для записи в log-файл
      d=new Date(Fil.DateCreated); // Выделение даты создания файла
      ss=LFillStr(13,Fil.name) // Выделение имени файла
      s1="("+DateToStr(d)+" "; // Формирование нужного представления даты
      s1+=TimeToStr(d)+")"; // Формирование нужного представления времени
      ss+=LFillStr(20,s1);
      d=new Date(); // В объекте d содержится текущая системная дата
      ss+=DateToStr(d); // Формирование нужного представления даты
      ss+=" "+TimeToStr(d); // Формирование нужного представления времени
      FLog.writeLine(ss); // Запись строки в log-файл
      }
    enumObj.moveNext();// Переход к следующему элементу коллекции
   }
}
/*************************************************/
/* Функция MoveFiles - перемещение файлов        */
/*************************************************/
function MoveFiles() {
var Col=0; // Переменная для подсчета количества перемещенных файлов
  // Вывод информации на экран
  WScript.Echo("");
  WScript.Echo("Перемещаем файлы...");
```

```
  // Создание объекта Enumerator для коллекции Fils
  enumObj=new Enumerator(Fils);
  while (!enumObj.atEnd()) { // Цикл для выделения нужных файлов
    Fil=enumObj.item();// Создание объекта File для элемента коллекции
    s=FSO.GetExtensionName(Source+Fil.name); // Выделение расширения фай-
ла
    if (s==Mask) { // Определение нужного файла
      WScript.Echo("  "+Fil.name); // Вывод на экран имени файла
      Fil.Copy(Dest); // Копирование файла
      Fil.Delete(); // Удаление файла
      Col++;
      }
    enumObj.moveNext(); // Переход к следующему элементу коллекции
    }
  // Вывод на экран информации о количестве перемещенных файлов
  WScript.Echo("");
  WScript.Echo(Col+" файла(ов) перемещен(о). Нажмите клавишу <Enter>");
  WScript.StdIn.ReadLine();
}
/*************************************************************/
/* Вспомогательные функции для формирования строк log-файла */
/*************************************************************/
function LeadZero(ll,ss) {
  var i,s,l1;
  s=ss.toString();
  l1=s.length;
  if (ll<=ll) {
    for (i=1;i<=ll-l1;i++)
      s="0"+s;
    }
  return(s);
  }
function DateToStr(dd) {
  var s;
  s=LeadZero(2,dd.getDate())+"/";
  s+=LeadZero(2,dd.getMonth()+1)+"/";
  s+=dd.getYear();
  return(s);
  }
function TimeToStr(dd) {
  var s;
  s=LeadZero(2,dd.getHours())+":"+LeadZero(2,dd.getMinutes());
  return(s);
  }
function LFillStr(l,s) {
var ss,i,ll;
```

```
ll=l-s.length;
if (s.length>=l) {
  return(s);
  }
else {
  ss=s;
  for (i=1;i<=ll;i++) {
    ss=ss+" ";
    }
  return(ss);
  }
}
```

Текст сценария снабжен подробными комментариями, поэтому мы лишь кратко остановимся на общем алгоритме его работы.

В начале сценария, после объявления переменных, создается коллекция `Fils`, содержащая все файлы каталога-источника, и открывается файл отчета в режиме добавления информации (объект `FLog`). Затем вызываются две основные функции: `WriteLog`, обеспечивающая вывод информации о перемещаемых файлах на экран и в файл отчета, и `MoveFiles`, которая выполняет перемещение нужных файлов. В этих функциях для выделения из каталога-источника необходимых файлов используется объект `Enumerator` с именем `enumObj`, позволяющий в цикле `while` просмотреть все элементы коллекции `Fils`. Для определения подходящих файлов из экземпляра коллекции `Fils` выделяется расширение файла и сравнивается с заданным (переменная `Mask`).

Вспомогательные функции `DateToStr`, `TimeToStr`, `LeadZero` и `LFillStr` предназначены для формирования строк файла отчета в требуемом формате:

```
43432_KL.485 (19/12/2000 10:45)  19/12/2000 11:02
```

В этих функциях производятся операции над строками и датами; для этой цели используются методы объектов `String` и `Date` языка JScript, описанные в гл. 5.

Доступ к базам данных из сценариев WSH

На практике довольно часто возникают задачи, для решения которых необходимо из сценариев получать доступ к данным, хранящимся во внешних базах самого различного формата (структурированные текстовые файлы, таблицы DBF и Paradox, базы данных Microsoft Access и Microsoft SQL Server и т. д.). Это довольно просто можно сделать с помощью технологии Microsoft ADO (ActiveX Data Objects). Объекты ADO являются частью *ком-*

понентов доступа к данным Microsoft (Microsoft Data Access Components, MDAC), которые поставляются, например, с браузером Microsoft Internet Explorer 5.0 или могут быть свободно получены с сервера Microsoft (**http://msdn.microsoft.com/data/download.htm**).

Мы не будем здесь останавливаться на объектной модели и принципах работы ADO (это слишком обширный вопрос, выходящий за рамки данной книги), а лишь рассмотрим несколько конкретных примеров использования данных из базы самой простой структуры: `dbf`-формата. Создать такую базу данных можно с помощью системы управления базами данных (СУБД) FoxPro, Microsoft Access или Microsoft Excel. В приведенных ниже сценариях будет использоваться файл `C:\ADO\names.dbf`, поля (столбцы) и записи (строки) которого представлены в табл. 6.14.

Таблица 6.14. *База данных* `names.dbf`

SName (фамилия)	Name (имя)	Wage (зарплата)
Иванов	Иван	1500.00
Петров	Борис	1100.00
Антонов	Павел	2000.00
Борисов	Антон	1600.00

Использование DSN и объекта *Recordset*

Разберем самый простой способ получения доступа к этой базе из сценария WSH с помощью ODBC (Open DataBase Connectivity). ODBC — это стандартное средство работы с реляционными базами данных от Microsoft, способное обрабатывать запросы к базам на языке SQL (Structured Query Language, язык структурированных запросов).

Вначале нам понадобится завести в системе ODBC-запись для связи с нашей базой, т. е. создать новый DSN (Data Source Name, имя источника данных). В Windows 9*x* это делается следующим образом (для Windows NT создание DSN производится аналогично).

1. Загрузите Панель управления Windows (меню **Пуск/Настройка**) и выберите там пункт **Источники данных ODBC**. В появившемся диалоговом окне выберите закладку **Системный DSN** (рис. 6.1).
2. Нажмите кнопку **Добавить** и в появившемся окне выберите драйвер Microsoft dBase Driver (*.dbf). После нажатия кнопки **Готово** появится новое окно, описывающее параметры подключения к нашей базе.
3. В поле **Имя источника данных** (Data Source Name) напишите имя, через которое мы будем осуществлять доступ к своей базе, к примеру,

NamesDSN. Для выбора пути к базе данных снимите флажок **Использовать текущий каталог** (Use Current Directory) и нажмите на кнопку **Выбор каталога** (Select Directory). В открывшемся окне укажите путь `C:\ADO` и нажмите кнопку **OK** (рис. 6.2).

4. Нажмите кнопку **OK** и закройте окно **Установка драйвера ODBC для dBASE**.

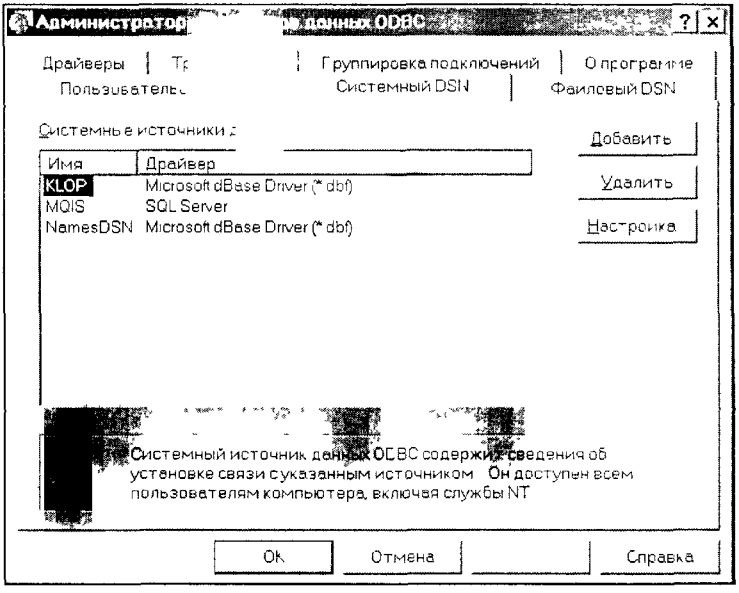


Рис. 6.1. Администратор источников данных ODBC в Windows 9*x*

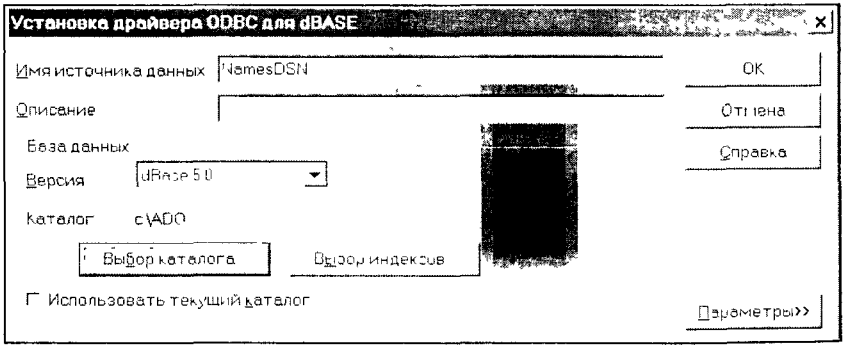


Рис. 6.2. Параметры источника данных ODBC

В листинге 6.10 приведен сценарий `printfields.js`, в котором с помощью объекта `Recordset`, представляющего собой таблицу-результат SQL-запроса к базе данных, обрабатываются данные из базы `names.dbf`.

Листинг 6.10. Вывод на экран значений всех полей базы names (файл printfields.js)

```
var rs, sSource, sConnect, field;
sSource = "SELECT * FROM names"; // SQL-запрос к таблице names.dbf
sConnect = "DSN=NamesDSN"; // Используемый DSN
rs=WScript.CreateObject("ADODB.Recordset"); // Создание объекта Recordset
rs.open(sSource, sConnect); // Открытие таблицы-результата запроса
WScript.Echo("ВСЕ ЗАПИСИ ИЗ ТАБЛИЦЫ names.dbf:");
while (!rs.EOF) {  // Цикл по всем записям таблицы rs
  // Печать значений полей текущей записи
  WScript.echo(rs.fields("Name"),rs.fields("SName"),rs.fields("Wage"));
  rs.moveNext(); // Переход к следующей записи
}
rs.Close(); // Закрытие объекта Recordset
```

Если запустить сценарий printfields.js с помощью cscript.exe, то на экран будут выведены все записи из файла names.dbf в порядке их размещения в базе:

```
ВСЕ ЗАПИСИ ИЗ ТАБЛИЦЫ names.dbf:
Иван Иванов 1500
Борис Петров 1100
Павел Антонов 2000
Антон Борисов 1600
```

Сделаем необходимые пояснения по работе сценария printfields.js. После создания объекта Recordset с помощью метода WScript.CreateObject ("ADODB.Recordset") мы используем метод open этого объекта для выполнения SQL-запроса (текст запроса записан в переменной sSource) к базе данных, DSN которой записан в строке sConnect (в нашем случае это NamesDSN). Оператор SQL SELECT * FROM names обеспечивает выбор из таблицы names всех записей; в результате объект rs будет содержать таблицу со всеми полями и записями из names.dbf.

Для перемещения по записям полученной таблицы используется так называемый *курсор*, который сначала устанавливается на первую запись. Перебор всех записей производится в цикле while, условием выхода из которого является перемещение курсора за последнюю запись таблицы (свойство EOF объекта Recordset). Для доступа к полям текущей записи используется коллекция fields, содержащая значения всех полей; чтобы получить значение конкретного поля, нужно указать в круглых скобках имя этого поля в кавычках, скажем, rs.fields("Name"). Метод moveNext выполняет переход к следующей записи таблицы. Закрывается объект Recordset с помощью метода Close.

Получение информации о полях и записях

Иногда бывает необходимо определить число и названия полей в таблице или подсчитать общее число записей Это можно сделать с помощью коллекции `fields`, содержащей все поля таблицы `Recordset`, и свойства `RecordCount`, в котором хранится число записей (листинг 6 11)

Листинг 6.11. Вывод на экран информации о полях базы names (файл columninfo.js)

```
var rs, sSource, sConnect, iField,nFields,nRecs, field;
sSource = "SELECT * FROM names"; // SQL-запрос к таблице names.dbf
sConnect = "DSN=NamesDSN"; // Используемый DSN
rs=WScript.CreateObject("ADODB.Recordset"); // Создание объекта Recordset
rs.CursorType = 3; // Задание статического курсора
rs.open(sSource, sConnect), // Открытие таблицы-результата запроса
nFields = rs.fields.count; // Определение числа полей в таблице rs
nRecs = rs.RecordCount; // Определение числа записей в таблице rs
WScript.Echo("ТАБЛИЦА names.dbf СОДЕРЖИТ",nFields,"ПОЛЯ(ЕЙ):");
// Цикл по всем полям
for (iField = 0; iField < rs.fields.count; iField++) {
  // Печать номера и имени текущего поля
  WScript.echo("Поле N" + iField + ":",rs.fields(iField).name);
}
WScript.Echo("И",nRecs,"ЗАПИСЬ(ЕЙ):"); // Печать числа записей
rs Close(); // Закрытие объекта Recordset
```

При выполнении сценарий `columninfo js` выведет на экран следующую информацию

```
ТАБЛИЦА names.dbf СОДЕРЖИТ 3 ПОЛЯ(ЕЙ):
Поле N0: SNAME
Поле N1: NAME
Поле N2: WAGE
И 4 ЗАПИСЬ(ЕЙ)
```

Отметим, что для доступа к свойству `RecordCount` необходимо перед открытием объекта `Recordset` установить свойство `CursorType` равным 3 (при этом создается статическая копия таблицы).

Сортировка и фильтрация данных

Используя предложение `ORDER BY` в операторе `SELECT`, можно получить из источника данных записи, отсортированные по заданному полю (листинг 6 12)

Листинг 6.12. Сортировка записей в базе names (файл ordname.js)

```
var rs, sSource, sConnect, field;
sSource="SELECT * FROM names ORDER BY SName"; // SQL-запрос для сортировки
sConnect = "DSN=NamesDSN", // Используемый DSN
rs=WScript.CreateObject("ADODB.Recordset"), // Создание объекта Recordset
rs.open(sSource, sConnect); // Открытие таблицы-результата запроса
WScript.Echo("СОРТИРОВКА ЗАПИСЕЙ ПО ФАМИЛИИ:");
while (!rs.EOF) { // Цикл по всем записям таблицы rs
  // Печать значений полей текущей записи
  WScript.echo(rs.fields("SName"),rs.fields("Name"),rs.fields("Wage"));
  rs.moveNext(); // Переход к следующей записи
}
rs.Close(), // Закрытие объекта Recordset
```

При выполнении сценарий ordname.js выводит информацию, упорядоченную по фамилиям работников

```
СОРТИРОВКА ЗАПИСЕЙ ПО ФАМИЛИИ.
Антонов Павел 2000
Борисов Антон 1600
Иванов Иван 1500
Петров Борис 1100
```

Предложение WHERE в операторе SELECT позволяет выделять только те записи, которые удовлетворяют определенному условию Например, приведенный в листинге 6 13 сценарий filter.js выводит отсортированную по фамилиям информацию только о тех работниках, зарплата которых превышает 1500 рублей

Листинг 6.13. Фильтрация записей в базе names (файл filter.js)

```
var rs, sSource, sConnect, iField, field, nRecs;
// SQL-запрос для выбора нужных данных и их сортировки
sSource = "SELECT * FROM names WHERE Wage>1500 ORDER BY SName";
sConnect = "DSN=NamesDSN"; // Используемый DSN
rs=WScript.CreateObject("ADODB.Recordset"); // Создание объекта Recordset
rs.CursorType = 3; // Задание статического курсора
rs.open(sSource, sConnect); // Открытие таблицы-результата запроса
nRecs = rs.RecordCount; // Определение числа записей в таблице rs
WScript.Echo("БОЛЬШЕ 1500 РУБ ПОЛУЧАЮТ:");
while (!rs.EOF) { // Цикл по всем записям таблицы rs
  // Печать значений полей текущей записи
  WScript.echo(rs.fields("SName"),rs.fields("Name"),rs.fields("Wage"));
  rs moveNext(); // Переход к следующей записи
}
```

```
WScript Echo("--------------------------------");
WScript.Echo("Всего: ",nRecs,"чел."); // Печать количества записей
rs.Close(); // Закрытие объекта Recordset
```

После выполнения сценария filter.js на экране мы получим такие строки

```
БОЛЬШЕ 1500 РУБ ПОЛУЧАЮТ:
Антонов Павел 2000
Борисов Антон 1600
--------------------------------
Всего:  2 чел.
```

Перемещение в наборе записей

В таблице Recordset можно перемещаться по записям не только вперед, но и назад Такое перемещение делается с помощью метода MovePrevious, для использования которого нужно установить тип курсора (свойство CursorType) в объекте Recordset равным 1, 2 или 3 (листинг 6 14)

Листинг 6.14. Перемещение в базе names (файл moveinbase.js)

```
var rs, sSource, sConnect, field,
sSource = "SELECT * FROM names"; // SQL-запрос к таблице names.dbf
sConnect = "DSN=NamesDSN"; // Используемый DSN
rs=WScript.CreateObject("ADODB.Recordset"); // Создание объекта Recordset
rs.CursorType = 3; // Задание статического курсора
rs.open(sSource, sConnect); // Открытие таблицы-результата запроса
WScript.Echo("ЗАПИСИ В ОБРАТНОМ ПОРЯДКЕ:");
rs.MoveLast(); // Переход на последнюю запись
while (!rs.BOF) { // Цикл по всем записям таблицы rs
  // Печать значений полей текущей записи
  WScript echo(rs.fields("Name"),rs.fields("SName"),rs.fields("Wage"));
  rs.MovePrevious(); // Переход к предыдущей записи
}
rs.Close(); // Закрытие объекта Recordset
```

Свойство BOF, используемое в цикле while, становится равным true, когда курсор будет находиться перед первой записью таблицы rs В результате выполнения сценария moveinbase.js на экран выведутся записи файла в обратном порядке

```
ЗАПИСИ В ОБРАТНОМ ПОРЯДКЕ:
Антон Борисов 1600
Павел Антонов 2000
Борис Петров 1100
Иван Иванов 1500
```

у объекта Recordset также имеется метод Move, позволяющий перемещаться в таблице на произвольную запись

Доступ к данным без создания DSN

Связываться с базами данных можно и без предварительного создания DSN (DSN-Less Database Access) Для этого необходимо в качестве второго параметра метода open объекта Recordset задать строку, в которой явно будут записаны название нужного драйвера и параметры соединения с базой Для используемой нами базы данных dbf-формата в качестве параметра достаточно указать каталог, в котором находятся нужные таблицы (DefaultDir) Соответствующий пример сценария приведен в листинге 6 15

Листинг 6.15. Доступ к базе names.dbf без использования DSN

```
var rs, sSource, sConnect, field;
sSource = "SELECT * FROM names"; // SQL-запрос к таблице names dbf
sConnect = "Driver={Microsoft dBase Driver (*.dbf)};DefaultDir=C:\\ADO";
rs=WScript CreateObject("ADODB.Recordset"); // Создание объекта Recordset
rs.open(sSource, sConnect); // Открытие таблицы-результата запроса
WScript.Echo("ВСЕ ЗАПИСИ ИЗ ТАБЛИЦЫ names.dbf:"),
while (!rs.EOF) {  // Цикл по всем записям таблицы rs
  // Печать значений полей текущей записи
  WScript.echo(rs.fields("Name"),rs.fields("SName"),rs.fields("Wage"));
  rs.moveNext(); // Переход к следующей записи
}
rs.Close(); // Закрытие объекта Recordset
```

К положительным моментам доступа к данным без предварительного создания DSN можно отнести то, что строка соединения с базой формируется во время выполнения сценария (имя базы данных можно передавать в качестве параметра) — это позволяет писать более гибкие сценарии Кроме этого, сценарий, не требующий предварительной настройки ODBC, легче переносить на другие машины Недостаток этого подхода состоит в невозможности установить контроль над соединением с базой в той мере, в какой это позволяет сделать ODBC (это важно при работе с базой данных, находящейся в сети)

Использование объекта *Connection*

Связь с базами и выборку из них данных можно производить не только с помощью метода open объекта Recordset, но и с помощью методов объекта Connection, который, собственно, и отвечает за связь приложения с базой данных В сценарии connect.js, приведенном в листинге 6 16, объект Connection используется для выборки всех записей из файла names dbf.

Листинг 6.16. Связь с базой names с помощью объекта Connection (файл connect.js)

```
var rs, sSource, sConnect, field,
sSource = "SELECT * FROM names"; // SQL-запрос к таблице names dbf
sConnect = "DSN=NamesDSN", // Используемый DSN
// Создание объекта Connection
cn = WScript CreateObject("ADODB Connection");
cn open(sConnect); // Открытие доступа к базе
rs=cn.Execute(sSource); // Выполнение SQL-запроса
WScript Echo("ВСЕ ЗАПИСИ ИЗ ТАБЛИЦЫ names dbf.");
while (!rs.EOF) {  // Цикл по всем записям таблицы rs
  // Печать значений полей текущей записи
  WScript echo(rs.fields("Name"),rs fields("SName"),rs.fields("Wage"));
  rs moveNext(), // Переход к следующей записи
}
cn.Close(); // Закрытие объекта Connection
```

Объект Connection в сценарии connect js создается с помощью метода CreateObject ("ADODB.Connection") объекта WScript Затем, используя метод open, мы устанавливаем соединение с базой данных, DSN которой записан в строке sConnect Метод execute выполняет подготовленный SQL-запрос (переменная sSource) и в качестве результата возвращает объект Recordset (объект rs) Дальше, как и во всех предыдущих примерах, мы работаем с объектом rs В последней строке сценария вызывается метод close объекта Connection, которыи не только разрывает соединение с базой данных, но и закрывает все объекты Recordset, связанные с этим соединением

Сценарии и объекты службы каталогов

Обсудим сначала термины "каталог" и "служба каталога", которые будут использоваться в этом разделе Под *каталогом* в общем смысле этого слова подразумевается источник информации, в котором хранятся данные о некоторых объектах Например, в телефонном каталоге хранятся сведения об абонентах телефонной сети, в библиотечном каталоге — данные о книгах, в каталоге фаиловой системы — информация о находящихся в нем файлах

Что касается компьютерных сетей (локальных или глобальных), здесь также уместно говорить о каталогах, содержащих объекты разных типов зарегистрированные пользователи, доступные сетевые принтеры и очереди печати и т д Для пользователей сети важно уметь находить и использовать такие объекты (а их в крупной сети может быть огромное количество), администраторы же сети должны поддерживать эти объекты в работоспособном состоянии Под *службой каталога* (directory service) понимается та часть рас-

пределенной компьютерной системы (компьютерной сети), которая предоставляет средства для поиска и использования имеющихся сетевых ресурсов Другими словами, служба каталога — это единое образование, объединяющее данные об объектах сети, и совокупность служб, осуществляющих манипуляцию этими данными

В гетерогенной (неоднородной) компьютернои сети могут одновременно функционировать несколько различных служб каталогов, например, NetWare Bindery для Novell Netware 3 x, NDS для Novell NetWaie 4 x/5 x, Windows Directory Service для Windows NT 4 0 или Active Diiectory для Windows 2000 Естественно, для прямого доступа к разным службам каталогов приходится использовать разные инструментальные средства, что усложняет процесс администрирования сети в целом Для решения этой проблемы можно применить технологию ADSI (Active Diiectoiy Service Interface) фирмы Microsoft, которая предоставляет набор объектов ActiveX, обеспечивающих единообразный, не зависящий от конкретного сетевого протокола, доступ к функциям различных каталогов Объекты ADSI поставляются с операционной системой Windows 2000, а также могут быть свободно получены с сервера Microsoft (**http://www.microsoft.com/NTWorkstation/Downloads/Other**)

Для того чтобы находить объекты в каталоге по их именам, необходимо определить для этого каталога *пространство имен* (namespace) Скажем, файлы на жестком диске находятся в пространстве имен файловой системы Уникальное имя файла определяется расположением этого файла в пространстве имен, например

```
C.\Windows\Command\command com
```

Пространство имен службы каталогов также предназначено для нахождения объекта по его уникальному имени, которое обычно определяется расположением этого объекта в каталоге, где он ищется Разные службы каталогов используют различные виды имен для объектов, которые они содержат ADSI определяет соглашение для имен, с помощью которых можно однозначно идентифицировать любой объект в гетерогенной сетевой среде Такие имена называются *строками связывания* (binding string) или *строками ADsPath* и состоят из двух частей Первая часть имени определяет, к какой именно службе каталогов мы обращаемся, например

`"LDAP://"`	—	для службы каталогов, созданной на основе протокола LDAP (Lightweight Directory Access Protocol), в том числе для Active Directory в Windows 2000,
`"WinNT·//"`	—	для службы каталогов Windows NT 4 0,
`"NDS·//"`	—	для службы каталогов NetWaie NDS (Novell Directory Service),
`"NWCOMPAT://"`	—	для службы каталогов NetWare Bindery

Вторая часть строки ADsPath определяет расположение объекта в конкретном каталоге. Приведем несколько примеров полных строк ADsPath:

```
"LDAP://ldapsrv1/CN=Kazakov,DC=DEV,DC=MSFT,DC=COM"
"WinNT://Domain1/Server1,Computer"
"WinNT://Domain1/Kazakov"
"NDS://TreeNW/O=SB/CN=Kazakov"
"NWCOMPAT://NWServer/MyNw3xPrinter"
```

В этом разделе мы подробно рассмотрим несколько простых сценариев, использующих объекты ADSI для автоматизации наиболее распространенных задач администрирования сети Windows NT 4.0. Напомним, что логическая структура такой сети представляется, как плоская модель доменов, которые могут быть связаны друг с другом доверительными отношениями. В каждом домене имеется один основной контроллер домена, на котором размещается база данных, содержащая информацию обо всех пользователях этого домена. Пользователи домена определяются своими атрибутами (имя регистрации, полное имя, пароль и т. п.) и могут объединяться в группы.

Ниже мы приведем примеры сценариев WSH, с помощью которых можно получить список имеющихся в сети доменов или всех групп определенного домена, добавить и удалить пользователя домена, определить всех пользователей заданной группы или все группы, в которые входит определенный пользователь. Также будет показано, каким образом можно просматривать атрибуты пользователя и изменять его пароль. Для получения более полной информации по технологии ADSI следует обратиться к документации Microsoft или специальной литературе *(см. введение)*.

Связывание с объектом сети Windows NT

Первым шагом для доступа к пространству имен любого каталога в целях получения информации о его объектах или изменения свойств этих объектов, является *связывание* (binding) с нужным объектом ADSI.

Рассмотрим вначале, каким образом формируется строка связывания для службы каталога Windows NT. В общем виде эта строка имеет следующий формат:

```
"WinNT:[//DomainName[/ComputerName[/ObjectName[,className]]]]"
```

Здесь параметр *DomainName* задает имя домена, *ComputerName* — имя сервера. *ObjectName* — имя объекта (это может быть имя группы, пользователя, сетевого принтера и т. п.). Так как в Windows NT 4.0 вполне могут встретиться объекты разных классов (например, пользователь и сетевой принтер) с одинаковыми именами, то в строке связывания предусмотрен параметр *className*, определяющий класс объекта. Возможными значениями этого параметра являются, например, group (группа пользователей), user (пользователь), printer (принтер) или service (сервис Windows NT).

Указав в качестве строки ADsPath просто "WinNT:", можно выполнить связывание с корневым объектом-контейнером, содержащим все остальные объекты сети.

Для связывания с объектом домена в строке связывания можно не указывать имя сервера:

```
"WinNT:[//DomainName[/ObjectName[,className]]]"
```

Можно также сразу задавать имя компьютера, не указывая, какому домену этот компьютер принадлежит. Для этого в строке связывания указывается имя класса computer:

```
"WinNT://ComputerName,computer"
```

Приведем несколько примеров строк связывания для доступа к различным объектам сети Windows NT 4.0 (табл. 6.15).

Таблица 6.15. *Варианты строк связывания в Windows NT 4.0*

Строка ADsPath	Описание
"WinNT:"	Строка для связывания с корневым объектом пространства имен
"WinNT://MyDomain"	Строка для связывания с доменом MyDomain
"WinNT://MyDomain/Kazakov,user"	Строка для связывания с пользователем Kazakov домена MyDomain
"WinNT://MyDomain/DC1/BankUsers, group"	Строка для связывания с группой BankUsers на контроллере домена DC1
"WinNT://Server1,computer"	Строка для связывания с компьютером Server1

Для того чтобы из сценария WSH использовать объект ADSI, соответствующий сформированной строке связывания, необходимо применить функцию GetObject языка JScript, которая возвращает ссылку на объект ActiveX, находящийся во внешнем каталоге. Например:

```
var NameSpaceObj = GetObject("WinNT:");
var DomainObj = GetObject("WinNT://MyDomain");
var UserObj = GetObject("WinNT://MyDomain/Kazakov,user");
var GroupObj = GetObject("WinNT://MyDomain/DC1/BankUsers, group");
var CompObj = GetObject("WinNT://Server1,computer");
```

Замечание

Во всех рассмотренных ранее сценариях для создания объектов ActiveX мы пользовались методами `CreateObject` и `GetObject` объекта `WScript` или объектом `ActiveXObject` языка JScript. Для связывания же с объектом ADSI нужно использовать именно функцию `GetObject` языка JScript (или VBScript)!

Перейдем теперь к рассмотрению конкретных примеров сценариев, использующих объекты ADSI.

Список всех доступных доменов

В листинге 6.17 приведен сценарий, в котором создается список названий всех доменов, доступных в сети, этот список хранится в переменной `List`. В рассматриваемом сценарии производятся следующие действия:

1. Создается корневой объект `NameSpaceObj` класса `Namespace` для сети Windows NT, который содержит все остальные объекты сети.
2. С помощью свойства `Filter` из коллекции `NameSpaceObj` выделяются все содержащиеся в ней объекты класса `Domain`.
3. Создается объект `E` типа `Enumerator` для доступа к элементам коллекции `NameSpaceObj`.
4. В цикле `while` выполняется перебор всех элементов коллекции, которые являются объектами класса `Domain`, название домена, хранящееся в свойстве `Name`, добавляется (вместе с символом разрыва строки) в переменную `List`.

Листинг 6.17. Вывод на экран списка всех доменов локальной сети

```
var NameSpaceObj,DomObj,E,List,
// Связывание с корневым объектом Namespace
NameSpaceObj = GetObject("WinNT:");
NameSpaceObj.Filter = Array("domain");
// Создание объекта Enumerator для доступа к коллекции NameSpaceObj
E=new Enumerator(NameSpaceObj);
List='';
while (!E.atEnd()) { // Цикл по всем элементам коллекции доменов
  DomObj=E.item(); // Извлечение элемента коллекции (объект класса Domain)
  List+=DomObj.Name+"\n", // Формирование строки с именами доменов
  E.moveNext(); // Переход к следующему элементу коллекции
}
// Вывод информации на экран
WScript.Echo("Все доступные домены");
WScript.Echo(List);
```

Создание пользователя и группы домена

Для создания пользователя следует выполнить следующие шаги (листинг 6 18)

1 Произвести связывание с нужным доменом, т е создать объект класса `Domain` (`DomainObj` в нашем примере)

2 Создать объект класса `User` для нового пользователя (`UserObj` в нашем примере) Для этого используется метод `Create` объекта класса `Domain`, в качестве параметров этого метода указывается имя класса "user" и имя создаваемого пользователя (в нашем примере это имя хранится в переменной `UserString`)

3 Вызвать метод `SetInfo` объекта класса `User`, этот метод сохраняет информацию о новом пользователе на основном контроллере домена

Листинг 6.18. Создание нового пользователя домена

```
var DomainObj, UserObj,
    UserStr = "XUser", // Имя создаваемого пользователя
DomainObj = GetObject("WinNT://DOMAIN1"); // Связывание с доменом Domain1
UserObj=DomainObj Create("user",UserStr), // Создание объекта класса User
UserObj SetInfo(), // Сохранение информации на сервере
```

Группа в домене создается аналогичным образом (листинг 6 19)

Листинг 6.19. Создание новой группы в домене

```
var DomainObj, GroupObj,
    GroupStr = "XGroup"; // Имя создаваемой группы
DomainObj = GetObject("WinNT://DOMAIN1"), // Связывание с доменом Domain1
// Создание объекта класса Group
GroupObj=DomainObj.Create("group", GroupStr);
GroupObj SetInfo();// Сохранение информации на сервере
```

Вывод информации о пользователе и смена его пароля

В листинге 6 20 приведен сценарий `userinfo js`, в котором мы производим связывание с пользователем `XUser` домена `Domain1` — создаем объект `UserObj`, в соответствующих полях которого хранится информация о пользователе Для изменения пароля применяется метод `SetPassword` объекта `UserObj` Еще раз напомним, что все произведенные изменения должны быть сохранены на основном контроллере домена с помощью метода `SetInfo`

Отметим, что сценарий userinfo.js должен запускаться с помощью команды cscript.exe, т к для организации диалога с пользователем используются стандартные входной (StdIn) и выходной (StdOut) потоки

Листинг 6.20. Вывод информации о пользователе домена и смена его пароля (файл userinfo.js)

```
var UserObj,s;
// Связывание с пользователем XUser домена Domain1
UserObj=GetObject("WinNT://DOMAIN1/XUser,user");
// Вывод на экран информации о пользователе
WScript.Echo('Имя ',UserObj.Name),
WScript.Echo('Описание.',UserObj Description);
WScript.Echo('----------------------------');
// Запрос на изменение пароля
WScript.StdOut Write('Изменить пароль (Y/N)?'),
s=WScript.StdIn.ReadLine();
if (s=='y'||s=='Y'){
  UserObj.SetPassword("NewPassword"); // Установка нового пароля
  UserObj.SetInfo(); // Сохранение информации на сервере
}
```

Удаление пользователя и группы домена

Пользователь и группа удаляются из домена так же просто, как и создаются — только вместо метода Create объекта класса Domain нужно вызвать метод Delete (листинг 6 21)

Листинг 6.21. Удаление пользователя и группы домена

```
var DomainObj,
    UserStr = "XUser", // Имя удаляемого пользователя
    GroupStr = "XGroup", // Имя удаляемой группы
DomainObj = GetObject("WinNT://DOMAIN1"); // Связывание с доменом Domain1
DomainObj.Delete("user", UserStr); // Удаление пользователя
DomainObj.Delete("group", GroupStr); // Удаление группы
```

Список всех групп домена

Принцип формирования списка всех групп заданного домена остается тем же, что и для формирования списка всех доступных доменов, однако первоначальное связывание нужно производить не с корневым объектом класса Namespace, а с нужным объектом класса Domain В приведенном в лис-

тинге 6 22 сценарии для связывания с доменом Domain1 мы создаем объект-контейнер DomainObj, в котором содержатся все объекты домена Domain1 Затем в цикле while выбираются лишь объекты класса Group, т е те объекты, у которых в поле Class записана строка Group

Листинг 6.22. Вывод на экран имен всех групп заданного домена

```
var DomainObj,E,List;
DomainObj = GetObject("WinNT.//DOMAIN1"), // Связывание с доменом Domain1
// Создание объекта Enumerator для доступа к коллекции DomainObj
E=new Enumerator(DomainObj);
List='',
while ('E atEnd()) { // Цикл по всем элементам коллекции объектов домена
  GroupObj=E item(), // Извлечение элемента коллекции
  if (GroupObj Class == "Group") // Выделение объектов класса Group
    List+=GroupObj.Name+"\n"; // Формирование строки с именами групп
  E.moveNext(); // Переход к следующему элементу коллекции
}
// Вывод информации на экран
WScript.Echo("Все группы в домене DOMAIN1:");
WScript Echo(List);
```

Список всех пользователей в группе

В листинге 6 23 приведен сценарий, в котором формируется список всех пользователей группы BankUsers домена Domain1 Для связывания с группой BankUsers создается объект GroupObj, коллекция пользователей этой группы создается с помощью метода Members Обработка всех элементов полученной коллекции осуществляется стандартным методом (объект Enumerator и цикл while)

Листинг 6.23. Вывод на экран имен всех пользователей заданной группы

```
var GroupObj,List,E,UserObj;
// Связывание с группой BankUsers домена Domain1
GroupObj=GetObject("WinNT.//Domain1/BankUsers,group");
// Создание объекта Enumerator для доступа к коллекции пользователей
E=new Enumerator(GroupObj.Members());
List='';
while ('E atEnd()) { // Цикл по всем элементам коллекции пользователей
  UserObj=E.item(); // Извлечение элемента коллекции класса User
  List+=UserObj Name+"\n"; // Формирование строки с именами пользователей
  E.moveNext(), // Переход к следующему элементу коллекции
}
```

```
// Вывод информации на экран
WScript.Echo("Все пользователи группы BankUsers в домене Domain1");
WScript.Echo(List);
```

Список всех групп, в которые входит пользователь

Для создания коллекции групп, членом которых является пользователь, необходимо выполнить связывание с нужным пользователем, т е создать объект класса `User`, и воспользоваться методом `Groups` этого объекта. Цикл по всем элементам коллекции, которые являются объектами класса `Group`, и выделение из этих элементов нужной информации производится так же, как в предыдущих примерах (листинг 6.24).

Листинг 6.24. Вывод на экран названия всех групп, членом которых является заданный пользователь

```
var UserObj,E,GroupObj;
// Связывание с пользователем Kazakov домена Domain1
UserObj = GetObject("WinNT://DOMAIN1/Kazakov");
// Создание объекта Enumerator для доступа к коллекции групп пользователя
E=new Enumerator(UserObj.Groups());
List='';
while (!E.atEnd()) { // Цикл по всем элементам коллекции групп
  GroupObj=E.item(); // Извлечение элемента коллекции класса Group
  List+=GroupObj.Name+"\n"; // Формирование строки с названиями групп
  E.moveNext(); // Переход к следующему элементу коллекции
}
// Вывод информации на экран
WScript.Echo("Пользователь Kazakov входит в группы:");
WScript.Echo(List);
```

Сценарии WSH как приложения XML

До сих пор мы рассматривали простые одиночные файлы сценариев, в которых мог использоваться язык JScript или VBScript В версии WSH 1 0 это был единственный поддерживаемый тип сценариев, причем используемый язык определялся по расширению файла· `js` для JScript и `vbs` для VBScript В WSH 2 0 появилась возможность создавать сценарии, в которых можно применять оба языка одновременно Для таких сценариев в операционной системе регистрируется расширение `wsf`; `wsf`-файлы мы будем далее называть просто `WS`-файлами Новыи тип сценариев (`WS`-файл) имеет еще не-

сколько важных преимуществ перед одиночными фаилами сценариев WSH 1 0

- поддерживаются вложенные файлы;
- возможен доступ из сценария к внешним мнемоническим константам, которые определены в библиотеках типов используемых объектов ActiveX;
- в одном WS-файле можно хранить несколько отдельных, независимых друг от друга, сценариев

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

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

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

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

Основные принципы XML

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

ных для отображения с помощью, например, языков сценариев JScript или VBScript.

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

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

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

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

Схема WS XML

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

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

Открывающий тег элемента состоит из следующих компонентов:

- открывающей угловой скобки "<";

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

Например, тег начала элемента

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

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

Закрывающий тег элемента состоит из следующих компонентов:

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

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

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

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

То есть в этом случае элемент состоит из следующих компонентов:

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

Пример такого элемента:

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

Представленная в табл. 6.16 схема WS XML — это модель данных, определяющая элементы и соответствующие атрибуты, а также связи элементов друг с другом и возможную последовательность появления элементов. Также схема может задавать значения атрибутов по умолчанию.

Таблица 6.16. *Схема WS XML*

```
<?XML version="1.0" standalone="yes"?>
    <package>
        <job [id="JobID"]>
            <object id="ObjID" [classId="clsid:GUID"|progid="ProgID"]/>
            <reference [object="ProgID"|guid="typelibGUID"][version="version"]/>
            <script language="language" [src="strFileURL"]\>
            <script language="language" >
                <![CDATA[
                Код сценария
                ]]>
            </script>
        </job>
        Другие задания
    </package>
```

Таким образом, элемент <package> может содержать один или несколько элементов <job>, а элемент <job> — один или несколько элементов <object>, <reference> или <script>. Обязательными для создания корректного сценария являются только элементы <job> и <script>. Сам код сценария всегда располагается внутри элемента <script>.

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

Элементы *WS*-файла

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

```
<!-- Первый комментарий -->
```

или

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

Элементы <?XML?> и <![CDATA[]]>

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

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

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

Замечание

В WSH 2.0 названия и значения атрибутов в элементе <?XML?> должны быть именно такими, как в табл. 6.16 (version="1.0" и standalone="yes").

Элемент <package>

Этот элемент необходим в тех WS-файлах, в которых с помощью элементов <job> определено более одного задания. В этом случае все эти задания должны находиться внутри пары тегов <package> и </package> (см. табл. 6.16). Если же в WS-файле определено только одно задание, то элемент <package> можно не использовать.

Элемент <job>

Элементы <job> позволяют определять несколько заданий (независимо выполняющихся частей) в одном WS-файле. Иначе говоря, между тегами <job> и </job> будет находиться отдельный сценарий.

У элемента <job> имеется единственный атрибут id, который определяет уникальное имя задания. Следующий схематичный пример определяет два задания с именами Task1 и Task2:

```
<job id="Task1">
   ...
</job>
```

```
<job id="Task2">
   ...
</job>
```

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

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

запускает с помощью cscript.exe задание с именем Task1 из файла main.wsf. По умолчанию из многозадачного WS-файла запускается первое задание.

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

Элемент <object>

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

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

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

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

или

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

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

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

или

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

Элемент <reference>

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

К счастью, большинство объектов предоставляет информацию об именах используемых ими констант в своей *библиотеке типов* (type library). Элемент `<reference>` как раз обеспечивает доступ к мнемоническим константам, определенным в библиотеке типов объекта (экземпляр объекта при этом не создается).

Для того чтобы воспользоваться константами определенного объекта, нужно в теге `<reference>` указать программный код этого объекта (атрибут `object`) или глобальный код его библиотеки типов (атрибут `guid`), а также, при необходимости, номер версии объекта (атрибут `version`). Например, доступ к константам объекта `FileSystemObject` организуется следующим образом:

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

После этого в сценариях можно просто использовать константы с именами `ForReading` или `ForAppending`, не заботясь об их числовых значениях (соответствующий пример приведен в листинге 6.26).

Элемент <script>

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

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

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

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

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

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

Элемент `<script>` является вторым обязательным элементом в сценариях WSH с разметкой XML

Примеры сценариев с разметкой XML

Приведем примеры сценариев, иллюстрирующие основные свойства `WS`-файлов

Строгий режим обработки *WS*-файла

Напомним, что здесь обязательными являются элементы `<?XML?>` и `<![CDATA[]]>` Например

```
<?XML version="1 0" standalone="yes" ?>
<job id="JS">
<script language="JScript">
<![CDATA[
  WScript.Echo("Всем привет!");
]]>
</script>
</job>
```

Несколько заданий в одном файле

Создадим сценарий `example.wsf`, приведенный в листинге 6 25

Листинг 6.25. Сценарий `example.wsf`

```
<package>
<job id="VBS">
<script language="VBScript">
   WScript.Echo "Первое задание (VBScript)"
</script>
</job>
```

```
<job id="JS">
<script language="JScript">
  WScript Echo("Второе задание (JScript)");
</script>
</job>
</package>
```

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

```
cscript //job:"VBS" example.wsf
cscript example.wsf
wscript //job:"VBS" example.wsf
cwcript example.wsf
```

Для запуска второго задания, выводящего на экран строку `Второе задание (JScript)`, используется одна из двух команд:

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

Использование констант внешних объектов

В листинге 6 26 приведен сценарий `refer.wsf`, в котором с помощью элемента `<reference>` производится доступ к константам объекта `FileSystemObject`.

Листинг 6.26. Использование в сценарии констант внешних объектов (файл `refer.wsf`)

```
<job id="Example">
<reference object="Scripting.FileSystemObject"/>
<script language="JScript">
  WScript.Echo("ForAppending="+ForAppending);
</script>
</job>
```

В результате выполнения сценария `refer.wsf` на экран выведется строка `ForAppending=8`

Подключение внешних файлов

Создадим файл `inc.js`, содержащий строку

```
WScript.Echo("Сценарий inc js");
```

и файл main.wsf следующего содержания:

```
<job id="Example">
<script language="JScript" src="inc.js"/>
<script language="JScript">
  WScript.Echo("Основной сценарий");
</script>
</job>
```

В результате выполнения WS-файла main.wsf выведутся две строки:

```
Сценарий inc.js
Основной сценарий
```

Два языка внутри одного задания

Напомним, что с помощью метода Popup объекта WshShell из сценариев JScript мы можем выводить информацию в окно с различными кнопками. Однако ни в WSH, ни в JScript нет метода или функции, которые позволяли бы создать диалоговое окно для ввода текста. К счастью, в языке VBScript имеется функция InputBox, предназначенная как раз для этой цели. Используя разметку XML, мы можем легко использовать эту функцию в сценариях JScript. Соответствующий пример приведен в листинге 6.27.

Листинг 6.27. Использование различных языков внутри одного задания (файл multi.wsf)

```
<job id="Example">
<script language="VBScript">
  Function InputName
  InputName = InputBox("Введите Ваше имя:", "Окно ввода VBScript")
  End Function
</script>
<script language="JScript">
    var s;
    s = InputName();
    WScript.Echo("Здравствуйте, "+s+"!");
</script>
</job>
```

После запуска сценария multi.wsf на экран выводится диалоговое окно для ввода имени пользователя (функция InputName на языке VBScript), показанное на рис. 6.3.

После ввода информации продолжается выполнение сценария JScript и на экран выводится окно, показанное на рис. 6.4.

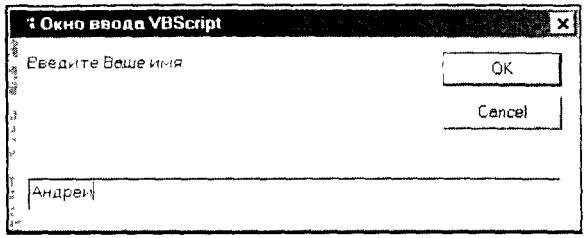


Рис. 6.3. Окно ввода (функция `InputBox` языка VBScript)

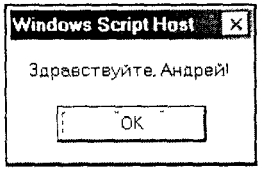


Рис. 6.4. Стандартное окно вывода WSH

Заключение

Приведенный в данной главе материал позволяет видеть, каким образом сценарии WSH применяются для решения следующих практических задач.

- *Обеспечение полного доступа к файловой системе компьютера.* Из сценария можно выполнять различные операции над объектами файловой системы:
  - получать сведения о файлах, каталогах и дисках;
  - создавать, копировать, перемещать и удалять файлы и каталоги;
  - получать списки всех доступных дисков или всех подкаталогов и всех файлов определенного каталога;
  - читать информацию из внешних текстовых файлов и записывать в них данные.
- *Обработка информации, хранящейся в базах данных различных типов.* С помощью технологии ADO (ActiveX Data Objects) и языка SQL (Structured Query Language) можно из сценариев WSH производить различные выборки из баз данных, изменять информацию в этих базах или записывать в них новые данные. Это позволяет, например, в сценариях регистрации использовать информацию о пользователях, хранящуюся во внешней базе данных.
- *Управление объектами локальных сетей.* Использование технологии ADSI (Active Directory Service Interface) дает возможность с помощью сценариев управлять учетными записями пользователей сети (одиночных или объединенных в группы), сетевыми принтерами и очередями печати и т. п. Это может значительно облегчить ежедневный труд администратора сети.

Отметим также, что с помощью сценариев WSH можно управлять работой приложений-серверов автоматизации (например, Microsoft Word или Microsoft Excel). Для работы с такими приложениями нужно знать используемую ими объектную модель (соответствующие примеры поставляются вместе с WSH).

Таким образом, для администратора операционной системы или компьютерной сети, знакомого с различными ActiveX-технологиями, WSH является чрезвычайно удобным и мощным инструментом, позволяющим создавать весьма сложные сценарии, автоматизирующие работу в Windows.

Упражнения

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

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

   должна сбросить этот атрибут у `example.txt`; если же первоначально файл не имел атрибута "Архивный", то его нужно установить. В сценарии должна быть предусмотрена проверка наличия указанного файла на диске.

2. Создать с помощью сценария текстовый файл `Отчет по дискам.txt`, в котором содержалась бы информация о дате и времени создания отчета, а также таблица использования дискового пространства на всех жестких дисках компьютера следующего формата:

Диск: *буква_диска*	
Метка тома: *метка*	**Общий объем, Mb:** *n1*
Используется, Mb: *n2*	**Свободно, Mb:** *n3*

3. Доработать сценарий, приведенный ранее в *разд. "Пример. Перемещение файлов с протоколированием действий"* данной главы, — перед перемещением файлов выполнять проверку доступности каталога, в который производится это перемещение.

4. Написать сценарий, выводящий на экран информацию о работающем пользователе: его имя регистрации в сети, список всех подключенных сетевых дисков (с указанием букв дисков и UNC-путей) и принтеров.

5. Вывести в информационное окно сведения из базы `C:\ADO\names.dbf` (структура и содержимое этой базы описаны ранее в *разд. "Доступ к базам данных из сценариев WSH"* данной главы) об одном работнике, фамилию которого ввести с помощью функции `InputBox` языка VBScript.

6. С помощью объектов ADSI сформировать из сценария текстовый файл, содержащий список всех доступных доменов сети Windows NT вместе с именами пользователей этого домена:

```
Домен: Domain1
  User1_1
  ...
  User1_M
Домен: Domain2
  User2_1
  ...
  User2_N
  ...
```

Заключение

В книге рассмотрены различные средства (от простейших командных файлов, присутствовавших еще в ранних версиях MS-DOS, до изощренных многозадачных сценариев WSH с разметкой XML), которые позволяют автоматизировать и существенно облегчить повседневную работу пользователей и администраторов операционных систем Windows 9*x*/NT/2000.

Любому пользователю, не говоря уже о системном администраторе, для грамотной работы просто необходимо уметь пользоваться командной строкой, четко знать возможности основных команд используемой версии Windows и понимать такие базовые механизмы функционирования операционной системы, как перенаправление ввода/вывода и конвейеризация команд. В книге подробно показано, каким образом можно применять командную строку в Windows 9*x* и Windows NT, какие именно команды используются в этих операционных системах для решения различных практических задач: манипулирования файлами и каталогами, поиска, сортировки и замены информации в файлах, использования ресурсов локальной сети, просмотра и редактирования переменных среды.

Специалистам, профессионально работающим с Windows NT, следует обратить особое внимание на утилиты командной строки из пакета Windows NT Resource Kit, которые способны упростить решение многих задач обслуживания системы и сэкономить время администратору.

Для быстрой и простой автоматизации повторяющихся рутинных задач идеальным средством являются пакетные (командные) файлы. В Windows 9*x* синтаксис таких файлов, полностью позаимствованный из MS-DOS, является очень простым, поэтому практически любой пользователь может быстро научиться самостоятельно писать нужные ему командные файлы. Однако такая идущая из MS-DOS простота командных файлов иногда является препятствием для людей, работающих с Windows NT, — они не обращают внимания на новые возможности и команды Windows NT, т. к. привыкли, что ничего более-менее серьезного сделать с помощью пакетного файла нельзя.

А ведь использование расширенного режима командного интерпретатора Windows NT позволяет, например, внутри командных файлов производить арифметические вычисления с переменными среды и подставляемыми параметрами, или считывать строки из внешнего текстового файла с возможностью выделения из них подстрок. Поэтому пользователям и особенно администраторам Windows NT и Windows 2000 было бы весьма полезно ознакомиться с возможностями командных файлов в этих операционных системах (это может пригодиться, в частности, при составлении сценариев входа пользователей).

Для того чтобы сделать следующий шаг в процессе автоматизации работы в Windows, следует освоить разработанный компанией Microsoft сервер сценариев Windows Scripting Host (WSH), с помощью которого можно создавать полноценные сценарии, работающие непосредственно в операционной системе и использующие внешние объекты ActiveX. Конечно, написание сценариев WSH, по сравнению с созданием командых файлов, поначалу может показаться довольно непростой задачей — кроме знания специальных языков сценариев (например, VBScript или JScript) и представления об объектах ActiveX, нужно знать, по крайней мере, собственную объектную модель WSH и структуру объекта `FileSystemObject`. Однако с помощью этих средств уже можно писать сценарии, которые имеют полный доступ к файловой системе компьютера, системному реестру и ресурсам локальной сети. Для использования продвинутых возможностей сценариев WSH (например, применение разных языков внутри одного сценария) необходимо изучить схему WS XML.

Основным назначением WSH является интеграция с помощью сценариев различных ActiveX-технологий компании Microsoft, предназначенных, например, для обеспечения доступа к базам данных (ActiveX Data Objects, ADO), службам каталогов (Active Directory Service Interface, ADSI), функциям отправки почты (Collaboration Data Objects, CDO) или для управления приложениями семейства Microsoft Office. В книге мы постарались дать общее представление о некоторых из этих технологий и привести практические примеры их использования из сценариев.

Итак, для составления грамотных и профессиональных сценариев WSH необходимо разобраться в нескольких смежных технологиях, однако затраченные усилия наверняка будут возграждены — с помощью ActiveX-сценариев можно быстро решать возникающие перед администратором операционной системы задачи практически любой сложности.

ПРИЛОЖЕНИЕ 1

Список команд DOS/Windows 9x/NT

В табл. П1.1 приведен перечень утилит командной строки в алфавитном порядке с указанием, в каких операционных системах Microsoft они применяются.

Таблица П1.1. *Команды DOS/Windows 9x/NT*

Команда	Операционная система			
	DOS 6.22	Windows 9x	Windows NT Workstation 4.0	Windows NT Server 4.0
APPEND	X		X	X
ARP		X	X	X
AT			X	X
ATTRIB	X	X	X	X
BACKUP			X	X
BREAK	X	X	X	X
BUFFERS	X	X	X	X
CACLS			X	X
CALL	X	X	X	X
CHCP	X	X	X	X
CHDIR (CD)	X	X	X	X
CHKDSK	X	X	X	X
CHOICE	X	X		
CLS	X	X	X	X

Таблица П1.1 (продолжение)

Команда	Операционная система			
	DOS 6.22	Windows 9x	Windows NT Workstation 4.0	Windows NT Server 4.0
CMD			X	X
COMMAND	X	X		
COMP			X	X
CONVERT			X	X
COPY	X	X	X	X
COUNTRY	X	X	X	X
CTTY	X	X		
DATE	X	X	X	X
DBLSPACE	X			
DEBUG	X	X	X	X
DEFRAG	X	X		
DEL (ERASE)	X	X	X	X
DELTREE	X	X		
DEVICE	X	X	X	X
DEVICEHIGH	X	X		
DIR	X	X	X	X
DISKCOMP	X		X	X
DISKCOPY	X	X	X	X
DOS	X	X	X	X
DOSKEY	X	X	X	X
DOSONLY			X	X
DOSSHELL	X			
DRIVPARM	X			
ECHO	X	X	X	X
ECHOCONFIG			X	X
EDIT	X	X	X	X
EDLIN			X	X
EMM386	X	X		

Таблица П1.1 (продолжение)

Команда	Операционная система			
	DOS 6.22	Windows 9*x*	Windows NT Workstation 4.0	Windows NT Server 4.0
ENDLOCAL			X	X
EXE2BIN			X	X
EXIT	X	X	X	X
EXPAND	X		X	X
FASTHELP	X			
FASTOPEN	X		X	X
FC	X	X	X	X
FCBS	X	X	X	X
FDISK	X	X		
FILES	X	X	X	X
FIND	X	X	X	X
FINDSTR			X	X
FINGER			X	X
FOR	X	X	X	X
FORMAT	X	X	X	X
FTP		X	X	X
GOTO	X	X	X	X
GRAFTABL			X	X
GRAPHICS	X		X	X
HELP	X		X	X
HOSTNAME			X	X
IF	X	X	X	X
INCLUDE	X	X		
INSTALL	X	X	X	X
INTERLNK	X			
INTERSVR	X			
IPCONFIG		X	X	X
KEYB	X	X	X	X

Таблица П1.1 (продолжение)

Команда	Операционная система			
	DOS 6.22	Windows 9x	Windows NT Workstation 4.0	Windows NT Server 4.0
LABEL	X	X	X	X
LASTDRIVE	X	X	X	X
LOADFIX	X	X	X	X
LOCK		X		
LOADHIGH (LH)	X	X	X	X
LPQ			X	X
LPR			X	X
MEM	X	X	X	X
MEMMAKER	X			
MENUCOLOR	X	X		
MENUDEFAULT	X	X		
MENUITEM	X	X		
MKDIR (MD)	X	X	X	X
MODE	X	X	X	X
MORE	X	X	X	X
MOVE	X	X	X	X
MSAV	X			
MSBACKUP	X			
MSCDEX	X	X		
MSD	X			
NBTSTAT		X	X	X
NET ACCOUNTS			X	X
NET CONFIG		X	X	X
NET COMPUTER				X
NET CONFIG SERVER			X	X
NET CONFIG WORKSTATION			X	X
NET CONTINUE			X	X

Таблица П1.1 (продолжение)

Команда	Операционная система			
	DOS 6.22	Windows 9x	Windows NT Workstation 4.0	Windows NT Server 4.0
NET DIAG		X		
NET FILE			X	X
NET GROUP				X
NET HELP		X	X	X
NET HELPMSG			X	X
NET INIT		X		
NET LOCALGROUP			X	X
NET LOGOFF		X		
NET LOGON		X		
NET NAME			X	X
NET PASSWORD			X	
NET PAUSE			X	X
NET PRINT		X	X	X
NET SEND			X	X
NET SESSION			X	X
NET START		X	X	X
NET STATISTICS			X	X
NET STOP			X	X
NET TIME		X	X	X
NET USE		X	X	X
NET USER			X	X
NET VER		X		
NET VIEW		X	X	X
NETSTAT		X	X	X
NLSFUNC	X	X	X	X
NTCMDPROMPT			X	X
NUMLOCK	X			
PATH	X	X	X	X

Таблица П1.1 (продолжение)

Команда	Операционная система			
	DOS 6.22	Windows 9x	Windows NT Workstation 4.0	Windows NT Server 4.0
PAUSE	X	X	X	X
PING		X	X	X
POWER	X			
PRINT	X			X
PROMPT	X	X	X	X
QBASIC	X		X	X
RCP			X	X
RECOVER			X	X
REM	X	X	X	X
RENAME (REN)	X	X	X	X
REPLACE	X		X	X
RESTORE	X		X	X
REXEC			X	X
RMDIR (RD)	X	X	X	X
ROUTE		X	X	X
RSH			X	X
SCANDISC	X	X		
SET	X	X	X	X
SETLOCAL			X	X
SETVER	X	X	X	X
SHARE	X		X	X
SHELL	X	X	X	X
SHIFT	X	X	X	X
SMARTDRV	X	X		
SORT	X	X	X	X
STACKS	X	X	X	X
START		X	X	X
SUBMENU	X	X		

Таблица П1.1 (окончание)

Команда	Операционная система			
	DOS 6.22	Windows 9*x*	Windows NT Workstation 4.0	Windows NT Server 4.0
SUBST	X	X	X	X
SWITCHES	X		X	X
SYS	X	X		
TFTP			X	X
TIME	X	X	X	X
TITLE			X	X
TRACERT		X	X	X
TREE	X		X	X
TYPE	X	X	X	X
UNDELETE	X			
UNFORMAT	X			
VER	X	X	X	X
VERIFY	X	X	X	X
VOL	X	X	X	X
VSAFE	X			
XCOPY	X	X	X	X

ПРИЛОЖЕНИЕ 2

Ошибки выполнения сценариев в WSH

Ошибки, которые могут возникнуть при выполнении сценариев WSH, *вместе с описанием возможных причин их появления, приведены в табл.* П2.1.

***Таблица П2.1.** Ошибки WSH 2.0*

Сообщение об ошибке	Причина
The shortcut path name must end with `.lnk` or `.url`	Попытка создать ярлык с неправильным расширением файла (расширение должно быть `lnk` или `url`)
There is no printer called ""`%1`""	Неправильно указано имя принтера при вызове метода `SetDefaultPrinter`
Unable to remove environment variable ""`%1`""	Вызов метода `Environment.Remove` для несуществующей переменной среды
Unable to remove registry key ""`%1`""	Вызов метода `RegDelete` для несуществующего ключа реестра
Invalid root in registry key ""`%1`""	Вызов метода `RegRead` или `RegWrite` для некорректного ключа реестра
Unable to open registry key ""`%1`"" for reading	Вызов метода `RegRead` для несуществующего ключа реестра
Unable to write to `wsh.log`. Please check with your administrator	При вызове метода `LogEvent` в Windows 9*x* файл `%windir%\wsh.log` оказался заблокированным для записи
Unable to save shortcut ""`%1`""	Попытка сохранить новый ярлык в файле, который уже существует и имеет атрибут "*Только для чтения*"

Таблица П2.1 *(окончание)*

Сообщение об ошибке	Причина
Protocol handler for ""%1"" failed to execute	Попытка установить ярлык на сетевой ресурс, использующий некорректно зарегистрированный обработчик протокола
Invalid syntax in URL: ""%1""	Сохранение ярлыка на сетевой ресурс, имеющий некорректный URL
Protocol handler for ""%1"" not found	Попытка установить ярлык на сетевой ресурс, использующий несуществующий обработчик протокола
Unable to execute - argument list is too long	Связано с запуском сценария при помощи технологии Drag-and-Drop: на файл сценария "опущено" слишком много параметров — имен файлов
Unable to wait for process	С помощью метода Run дано указание ожидать завершение процесса, которое из сценария определить нельзя

ПРИЛОЖЕНИЕ 3

Различия в версиях WSH

На момент написания книги было доступно три версии Windows Scripting Host: WSH 1.0, WSH 2.0 и бета-версия WSH 5.6. В табл. П3.1 показано, какие версии WSH входят в поставку различных версий Microsoft Windows.

Таблица П3.1. *WSH как компонент поставки Microsoft Windows*

Операционная система	Версия WSH	
	WSH 1.0	WSH 2.0
Windows 98	X	
Windows NT 4.0 Option Pack	X	
Windows 2000		X

Кратко опишем возможности и особенности каждой из версий WSH.

Особенности WSH 1.0

Сценарии могут запускаться в текстовом (консольном) или в графическом режимах и представляют собой одиночные файлы с расширениями `js` (язык Microsoft JScript) или `vbs` (язык Microsoft VBScript). С помощью сценариев WSH 1.0 можно выполнять следующие основные функции:

- использовать внешние объекты ActiveX (в том числе `FileSystemObject` для работы с файловой системой);
- запускать другие процессы;
- выводить информацию в текстовом либо графическом режимах;

- получать доступ к специальным папкам Windows;
- создавать ярлыки Windows или изменять свойства уже существующих ярлыков;
- добавлять, изменять и удалять переменные окружения;
- читать и записывать данные системного реестра Windows;
- использовать ресурсы локальной сети (подключать или отключать сетевые диски, принтеры и выполнять другие действия).

Дополнительные возможности WSH 2.0

Сценарии WSH 2.0 можно запускать с помощью технологии "Drag-and-Drop": в проводнике можно выделить один или несколько файлов и перетащить их на ярлык файла со сценарием. При этом имена файлов рассматриваются как параметры командной строки, и сценарий запускается с этими параметрами.

Кроме одиночных `js`- и `vbs`-файлов, сценарии WSH 2.0 могут быть записаны в файлах с расширением `wsf` (`ws`-файлах), которые являются приложениями XML (Extensible Markup Language). Новый формат сценариев обеспечивает следующие возможности:

- поддержку вложенных файлов;
- доступ из сценария к внешним мнемоническим константам, которые определены в библиотеках типов используемых объектов ActiveX;
- использование внутри одного сценария функций, написанных на разных языках;
- возможность хранения в одном `ws`-файле несколько отдельных, независимых друг от друга, сценариев.

Также существуют следующие новые возможности сценариев, не связанные с разметкой XML:

- приостановка выполнения сценария с помощью нового метода `Sleep` объекта `WScript`;
- доступ из сценария к стандартному входному/выходному потоку (`StdIn/StdOut`) и стандартному потоку для ошибок (`StdErr`);
- активизация уже запущенного приложения с помощью метода `AppActivate` объекта `WshShel`;
- посылка нажатия клавиш в активное окно Windows с помощью метода `SendKeys` объекта `WshShell`.

Новые возможности WSH 5.6

В WSH 5.6 доработана схема WS XML и добавлено несколько новых объектов (`WshController`, `WshNamed`, `WshRemote`, `WshScriptExec`, `WshUnnamed` и др.), что позволяет:

- производить более удобную и гибкую обработку аргументов сценария;
- запускать порожденные процессы (приложения командной строки) с возможностью доступа к стандартным потокам (`StdIn/StdOut/StdErr`) и получения информации о ходе выполнения этих процессов;
- удаленно запускать сценарии на других машинах с возможностью использования объектов ActiveX, зарегистрированных на этих машинах, и получения информации о ходе выполнения запущенных сценариев (это позволяет с помощью сценариев удаленно администрировать рабочие станции и серверы в локальной сети);
- использовать новую модель безопасности при применении сценариев WSH, с помощью которой:
  - разработчик может произвести цифровое подписывание сценария для предотвращения его несанкционированного изменения;
  - администратор операционной системы — определить ограничения на запуск сценариев для пользователей;
  - пользователь — проверить корректность сценария перед его использованием.

В табл. П3.2 приведены сведения о наличии того или иного элемента WSH (объекта, свойства, метода или тега XML) в различных версиях сервера сценариев. В эту таблицу не включены свойства и методы объекта `FileSystemObject` и других объектов, предназначенных для работы с файловой системой.

Таблица П3.2. *Наличие элементов WSH в различных версиях*

Название элемента	Тип элемента	WSH 1.0	WSH 2.0	WSH 5.6
`<?job ?>`	Тег XML		X	X
`<?XML ?>`	Тег XML		X	X
`AddPrinterConnection`	Метод	X	X	X
`AddWindowsPrinterConnection`	Метод		X	X
`AppActivate`	Метод		X	X
`Arguments`	Свойство	X	X	X

Таблица П3.2 (продолжение)

Название элемента	Тип элемента	WSH 1.0	WSH 2.0	WSH 5.6
AtEndOfLine	Свойство		X	X
AtEndOfStream	Свойство		X	X
Character	Свойство			X
Close	Метод		X	X
Column	Свойство		X	
ComputerName	Свойство	X	X	X
ConnectObject	Метод		X	X
Count	Метод	X	X	X
CreateObject	Метод	X	X	X
CreateScript	Метод			X
CreateShortcut	Метод	X	X	X
CurrentDirectory	Свойство			X
Description	Свойство	X	X	X
Description (WshRemote)	Свойство			X
DisconnectObject	Метод	X	X	X
Echo	Метод	X	X	X
EnumNetworkDrives	Метод	X	X	X
EnumPrinterConnections	Метод	X	X	X
Environment	Свойство	X	X	X
Error (WshRemote)	Свойство			X
<example>	Тег XML			X
Exec	Метод			X
Execute	Метод			X
Exists	Метод			X
ExitCode	Свойство			X
ExpandEnvironmentStrings	Метод	X	X	X
FullName	Свойство	X	X	X
GetObject	Метод	X	X	X
getResource	Метод		X	X

Таблица П3.2 (продолжение)

Название элемента	Тип элемента	WSH 1.0	WSH 2.0	WSH 5.6
HotKey	Свойство	X	X	X
IconLocation	Свойство	X	X	X
Item	Свойство	X	X	X
Item (WshNamed)	Свойство			X
Item (WshUnnamed)	Свойство			X
<job>	Тег XML		X	X
Length	Свойство	X	X	X
Line	Свойство		X	X
Line (WshRemote)	Свойство			X
LogEvent	Метод		X	X
MapNetworkDrive	Метод	X	X	X
Name	Свойство	X	X	X
<named>	Тег XML			X
Number	Свойство			X
<object>	Тег XML		X	X
<package>	Тег XML		X	X
Path	Свойство	X	X	X
Popup	Метод	X	X	X
ProcessID	Свойство			X
Quit	Метод	X	X	X
Read	Метод		X	X
ReadAll	Метод		X	X
ReadLine	Метод		X	X
<reference>	Тег XML		X	X
RegDelete	Метод	X	X	X
RegRead	Метод	X	X	X
RegWrite	Метод	X	X	X
Remove	Метод	X	X	X
RemoveNetworkDrive	Метод	X	X	X

Таблица П3.2 (продолжение)

Название элемента	Тип элемента	WSH 1.0	WSH 2.0	WSH 5.6
RemovePrinterConnection	Метод	X	X	X
<resource>	Тег XML		X	X
Run	Метод	X	X	X
<runtime>	Тег XML			X
Save	Метод	X	X	X
<script>	Тег XML		X	X
ScriptFullName	Свойство	X	X	X
ScriptName	Свойство	X	X	X
SendKeys	Метод		X	X
SetDefaultPrinter	Метод	X	X	X
ShowUsage	Метод			X
Skip	Метод		X	X
SkipLine	Метод		X	X
Sleep	Метод		X	X
Source Property	Свойство			X
SourceText Property	Свойство			X
SpecialFolders Property	Свойство	X	X	X
Status (WshRemote)	Свойство			X
Status (WshScriptExec)	Свойство			X
StdErr	Свойство		X	X
StdErr (WshScriptExec)	Свойство			X
StdIn	Свойство		X	X
StdIn (WshScriptExec)	Свойство			X
StdOut	Свойство		X	X
StdOut (WshScriptExec)	Свойство			X
TargetPath	Свойство	X	X	X
Terminate (WshScriptExec)	Метод			X
UserDomain	Свойство	X	X	X
UserName	Свойство	X	X	X

Таблица П3.2 (окончание)

Название элемента	Тип элемента	WSH 1.0	WSH 2.0	WSH 5.6
Version	Свойство	X	X	X
WindowStyle	Свойство	X	X	X
WorkingDirectory	Свойство	X	X	X
Write	Метод		X	X
WriteBlankLines	Метод		X	X
WriteLine	Метод		X	X
WScript	Объект	X	X	X
WshArguments	Объект	X	X	X
WshController	Объект			X
WshEnvironment	Объект	X	X	X
WshNamed	Объект			X
WshNetwork	Объект	X	X	X
WshRemote	Объект			X
WshRemoteError	Объект			X
WshScriptExec	Объект			X
WshShell	Объект	X	X	X
WshShortcut	Объект	X	X	X
WshSpecialFolders	Объект	X	X	X
WshUnnamed	Объект			X
WshUrlShortcut	Объект	X	X	X

ПРИЛОЖЕНИЕ 4

Ответы к упражнениям

Приведенные ниже ответы к упражнениям представляют собой полностью готовые к исполнению командные файлы и сценарии, которые снабжены краткими комментариями, объясняющими их работу.

Упражнения главы 2

1. Практически все нужные действия выполняются с помощью конвейеризации команд FIND и SORT:

```
@ECHO OFF
REM Проверка наличия параметров командной строки
IF -%1==- GOTO NoParam
IF -%2==- GOTO NoParam
REM Выделение нужных строк из файла protokol.txt
FIND "%1.%2" protokol.txt | SORT /+26 > %1%2.txt
GOTO End

:NoParam
ECHO Не заданы необходимые параметры командной строки!
PAUSE
:End
```

2. Для выполнения поставленной задачи можно перебрать в цикле все файлы с расширением txt, проверяя перед копированием имя каждого из этих файлов:

```
@ECHO OFF
REM Проверка наличия параметра командной строки
IF -%1==- GOTO NoDir
```

```
REM Копирование нужных файлов
FOR %%f IN (*.txt) DO IF NOT -%%f==-%2 COPY %%f %1
GOTO End

:NoDir
ECHO Не указан каталог для копирования!
PAUSE
:End
```

3. В этом упражнении нужно создать два пакетных файла. В основном файле работает такой же цикл `FOR`, как и в предыдущем упражнении, однако вместо непосредственного выполнения команды `COPY` здесь вызывается командный файл `3_1.bat`:

```
@ECHO OFF
REM Проверка наличия параметра командной строки
IF -%1==- GOTO NoDir
REM Вызов в цикле файла 3_1.bat для копирования нужного файла
FOR %%f IN (*.txt) DO IF NOT -%%f==-%2 CALL 3_1.bat %%f %1
GOTO End

:NoDir
ECHO Не указан каталог для копирования!
PAUSE
:End
```

Вызываемый в цикле файл `3_1.bat` имеет следующее содержимое:

```
@ECHO OFF
REM Копирование файла
XCOPY %1 %2 /D /C > NUL
REM Проверка успешности копирования
IF ERRORLEVEL 0 GOTO Success
REM Запись в файл отчета информации об ошибке при копировании
ECHO Ошибка: %1 >> logcopy.log
GOTO End
:Success
REM Запись в файл отчета информации об успешном копировании
ECHO Успешно: %1 >> logcopy.log
:End
```

4. Следующий простой пакетный файл не требует дополнительных пояснений:

```
@ECHO OFF
CLS
IF -%1==- GOTO NoDir
```

```
REM Вывод меню на экран
ECHO A  - На экран
ECHO Б  - В файл C:\catalog.txt
ECHO В  - На принтер
REM Вывод подсказки для ввода
CHOICE /C:АБВ  Куда выводить содержимое %1
CLS
REM Определение сделанного выбора
IF ERRORLEVEL 3 GOTO DirToPrn
IF ERRORLEVEL 2 GOTO DirToFile
IF ERRORLEVEL 1 GOTO DirToCon
ECHO Выбор не был сделан.
GOTO End
:DirToCon
DIR %1 | MORE
GOTO End
:DirToFile
DIR %1 > C:\catalog.txt
GOTO End
:DirToPrn
DIR %1 > prn
GOTO End

:NoDir
ECHO Не указан каталог для сканирования!
PAUSE
:End
```

Упражнения главы 4

1. Необходимые комментарии приведены в тексте следующего командного файла:

```
@ECHO OFF
CLS
SET MyVar=0
REM Подсчет количества переменных среды
FOR /F %%i IN ('SET') DO CALL :Inc
ECHO Количество переменных в системе: %MyVar%
PAUSE
SET MyVar=0
REM Вывод имен переменных на экран
FOR /F "DELIMS== TOKENS=1,2" %%i IN ('SET') DO @CALL :PrintVars %%i %%j
GOTO :EOF
```

```
:Inc
SET /A MyVar=MyVar+1
GOTO :EOF

:PrintVars
SET /A MyVar=MyVar+1
ECHO %MyVar%. %1
```

2. От предыдущего упражнения следующий пакетный файл отличается большим количеством проверок (и, соответственно, большим количеством меток):

```
@ECHO OFF
CLS
REM Проверка наличия параметров командной строки
IF -%1==- GOTO ListHelp
IF NOT -%2==- GOTO Rewr

:Cont
SET MyVar=0
REM Подсчет количества переменных среды
FOR /F %%i IN ('SET') DO CALL :Inc
ECHO Количество переменных в системе: %MyVar%
PAUSE
GOTO :List

:ListHelp
ECHO "Синтаксис: listvar.bat [/A|/B] [имя_файла]"
GOTO :EOF

:Rewr
REM Очистка файла, в который выводится информация
ECHO. > %2
GOTO Cont

:Inc
SET /A MyVar=MyVar+1
GOTO :EOF

:List
SET MyVar=0
FOR /F "DELIMS== TOKENS=1,2" %%i IN ('SET') DO @CALL :PrVar %%i %%j %1 %2
IF NOT -%2==- START notepad %2
GOTO :EOF

:PrVar
SET /A MyVar=MyVar+1
```

```
IF -%4==- GOTO ToCon
REM Вывод в файл
IF %3==/A GOTO FullFile
REM Вывод только имени переменной
ECHO %MyVar%. %1 >> %4
GOTO :EOF

:FullFile
REM Вывод имени переменной вместе с ее значением
ECHO %MyVar%. %1   (%2) >> %4
GOTO :EOF

:ToCon
REM Вывод на экран
IF %3==/A GOTO FullCon
REM Вывод только имени переменной
ECHO %MyVar%. %1
GOTO :EOF

:FullCon
REM Вывод имени переменной вместе с ее значением
ECHO %MyVar%. %1   (%2)
GOTO :EOF
```

3. Приведем пример файла, который удалит все подкаталоги, размер которых превышает 20 Мбайт, в каталоге `D:\Profiles`:

```
@ECHO OFF
DIRUSE /S /M /Q:20 /L /* D:\Profiles
REM Выделение из файла diruse.log всех строк, содержащих пути
REM к каталогам
FIND ":\" diruse.log > 123.txt
REM Удаление всех подкаталогов, пути к которым записаны в 123.txt
FOR /F "TOKENS=1,2*" %%i IN (123.txt) DO RMDIR /S /Q %%j
```

4. В следующем примере мы считаем, что файл `mes.txt` с сообщениями для пользователей находится в каталоге `\\Server1\Mes`:

```
@ECHO OFF
FOR /F "TOKENS=1*" %%i IN (\\Server1\Mes\mes.txt) DO CALL :Check %%i %%j
GOTO :EOF

:Check
REM Проверка, тому ли пользователю адресовано сообщение?
IF %USERNAME%==%1 GOTO PrintMes
GOTO :EOF
```

```
:PrintMes
REM Вывод сообщения на экран
ECHO %*
REM Пауза 30 секунд
TIMEOUT 30
```

5. Приведенный ниже сценарий входа в комментариях не нуждается:

```
@ECHO OFF
NET TIME \\Server1 /SET
NET USE M: \\Server1\Letters /PERSISTENT:NO
NET SHARE MyTxt=C:\TEXT
```

6. Все необходимые комментарии приведены в тексте следующего пакетного файла:

```
@ECHO OFF
ECHO. > ss.txt
REM Проверка наличия параметров командной строки
IF -%2==- GOTO NoParam
REM Выделение фамилий и сумм из файла sums.txt
FOR /F "TOKENS=1-4 DELIMS=|" %%i IN (sums.txt) DO CALL :CheckSum
%%i %%l %1 %2
ECHO Диапазон сумм: от %1 до %2 > suminfo.txt
REM Добавление отсортированных строк в suminfo.txt
SORT < ss.txt >> suminfo.txt
ECHO Файл suminfo.txt сформирован!
PAUSE
GOTO :EOF

:NoParam
ECHO Не указан диапазон поиска!
PAUSE
GOTO :EOF

:CheckSum
REM Проверка попадания суммы в диапазон и запись информации в ss.txt
IF %2 LSS %3 GOTO :EOF
IF %2 LEQ %4 ECHO %1 %2 >> ss.txt
```

Упражнения главы 6

1. В функции `IsFileExists` проверяется наличие файла, заданного в качестве параметра командной строки. В функции `ToggleArchiveBit` производится переключение атрибута "Архивный" у этого файла.

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

function IsFileExists() {
  var objArgs = WScript.Arguments;
  if (objArgs.Count()==0) {
    WScript.Echo("Не задано имя файла!");
    return false;
    }
  else
    FileName = objArgs(0);
    if (!fso.FileExists(FileName)) {
      WScript.Echo("Файл",FileName,"не существует");
      return false;
    }
    else
     return true;
  }

function ToggleArchiveBit() {
  var f;
  f = fso.GetFile(FileName)
  if (f.attributes & 32) {
    f.attributes = f.attributes - 32;
    s = 'Файл '+FileName+': атрибут "Архивный" снят.';
  }
  else {
    f.attributes = f.attributes + 32;
    s = 'Файл '+FileName+': атрибут "Архивный" установлен.';
  }
  return(s);
  }

if (IsFileExists())
  WScript.Echo(ToggleArchiveBit());
```

2. Основными в сценарии являются функции `LoopDrives`, в которой в цикле перебираются все доступные диски компьютера, и `WriteDriveInfo`, в которой производится вывод в файл `Отчет по дискам.txt` информации в требуемом формате об одном диске.

```
var fso, rf, mdate,
    ForWriting = 2;
fso = new ActiveXObject("Scripting.FileSystemObject");
rf = fso.OpenTextFile("Отчет по дискам.txt", ForWriting, true );
mdate = new Date();
```

```
rf.WriteLine("Дата отчета: " + mdate);
rf.WriteLine();
LoopDrives();
rf.Close();
WScript.Echo('Файл "Отчет по дискам.txt" создан');

/*****************************************************/
/*   Перебор всех устройств (дисков)                 */
/*****************************************************/
function LoopDrives() {
  var e;
  /* Создание коллекции дисков */
  e = new Enumerator( fso.Drives );
  /* Цикл по всем дискам */
  for(; !e.atEnd(); e.moveNext())
    WriteDriveInfo(e.item());
  }

/*****************************************************/
/*   Вывод информации об одном устройстве (диске)    */
/*****************************************************/
function WriteDriveInfo(drive) {
var s, Total, Free, Used;
  if ( drive.IsReady ) { // Устройство готово
   Total = Math.round(drive.TotalSize/1048576 );
   Free = Math.round(drive.FreeSpace/1048576 );
   Used = Total - Free;
   rf.WriteLine("+-------------------------------------------+");
   s="|"+FillStr(51,"Диск "+drive.DriveLetter+":")+"|";
   rf.WriteLine(s);
   rf.WriteLine("+-------------------------------------------+");
   s="|"+LFillStr(25,"Метка тома: "+drive.VolumeName)+"|";
   s+=LFillStr(25,"Общий объем, Mb: "+Total)+"|";
   rf.WriteLine(s);
   rf.WriteLine("+-------------------------------------------+");
   s="|"+LFillStr(25,"Используется, Mb: "+Used.toString())+"|";
   s+=LFillStr(25,"Свободно, Mb: "+Free.toString())+"|";
   rf.WriteLine(s);
   rf.WriteLine("+-------------------------------------------+");
   rf.WriteLine();
   rf.WriteLine();
   }
  else
   if ( drive.DriveType == 3 ) { // Сетевой диск
     rf.WriteLine( "Диск " + drive.DriveLetter + " является сетевым" );
```

```
      rf.WriteLine();
      rf.WriteLine();
      }
    else { // Устройство не готово
      rf.WriteLine( "Устройство " + drive.DriveLetter + ": не готово"
);
      rf.WriteLine();
      rf.WriteLine();
      }
    }

/*************************************************************/
/*   Вспомогательные функции                                 */
/*************************************************************/
function RFillStr(l,s) {
  var ss,i,ll;
  ll=l-s.length;
  if (s.length>=l) {
    return(s);
    }
  else {
    ss=s;
    for (i=1;i<=ll;i++) {
      ss=" "+ss;
      }
    return(ss);
    }
  }

function LFillStr(l,s) {
  var ss,i,ll;
  ll=l-s.length;
  if (s.length>=l) {
    return(s);
    }
  else {
    ss=s;
    for (i=1;i<=ll;i++) {
      ss=ss+" ";
      }
    return(ss);
    }
  }

function FillStr(l,s) {
  var ss,i,ll,s1,l2;
```

```
  ll=l-s.length;
  if (s.length>=l) {
    return(s);
    }
  else  {
    ss=s;
    l2=Math.round((l-s.length)/2);
    ss=LFillStr(s.length+l2,s);
    ss=RFillStr(l,ss);
    return(ss);
    }
  }
```

3. Приведем только ту часть сценария, в которой производится требуемая проверка:

```
FSO=WScript.CreateObject("Scripting.FileSystemObject");
if (!FSO.FolderExists(Source)) {
  WScript.Echo("Каталог ",Source,"недоступен. Нажмите Enter...");
  WScript.StdIn.ReadLine();
  }
else
  if (!FSO.FolderExists(Dest)) {
    WScript.Echo("Каталог ",Dest,"недоступен. Нажмите Enter...");
    WScript.StdIn.ReadLine();
    }
  else {
    Fold=FSO.GetFolder(Source);
    Fils=Fold.Files;
    FLog=FSO.OpenTextFile(PathLog,ForAppending,true);
    WriteLog();
    MoveFiles();
    FLog.Close();
    }
```

4. Следующий простой сценарий практически не требует пояснений:

```
var objNet,e,i;
objNet = WScript.CreateObject("WScript.Network");
WScript.Echo("Имя пользователя:",objNet.UserName);
WScript.Echo("Подключенные диски:");
e = objNet.EnumNetworkDrives();
i = 0;
while (i<=e.Count()-1) {
  WScript.Echo("   Буква:", e.Item(i), "   Путь:", e.Item(i+1));
  i+=2;
  }
```

```
WScript.Echo("Подключенные принтеры:");
e = objNet.EnumPrinterConnections();
i = 0;
while (i<=e.Count()-1) {
  WScript.Echo("   Порт:", e.Item(i), "   Принтер:", e.Item(i+1));
  i+=2;
  }
```

5. Для выполнения поставленной задачи нужно в сценарии использовать разметку XML:

```
<job id="Example">
<script language="VBScript">
  Function InputName
  InputName=InputBox("Введите имя и фамилию работника:", "Поиск данных")
  End Function
</script>

<script language="JScript">
  var rs,s,ArrS;
  var sSource, sConnection, iField, field, nRecs;
  var WshShell=WScript.CreateObject("WScript.Shell");

  s = InputName();
  // Выделение из строки s фамилии и имени
  ArrS = s.split(" ");
  SName = ArrS[1]; // Фамилия
  Name =  ArrS[0]; // Имя
  // Построение соответствующего SQL-запроса
  sSource = "SELECT * FROM names WHERE (Name=\'";
  sSource += Name+"\' AND SName=\'"+SName+"\')";
  sConnection = "DSN=NamesDSN";
  rs = WScript.CreateObject("ADODB.Recordset");
  rs.open(sSource,sConnection);
  if (rs.RecordCount!=0) {
    s="Имя: "+rs.fields("Name")+"\n";
    s+="Фамилия: "+rs.fields("SName")+"\n";
    s+="Зарплата: "+rs.fields("Wage")+" руб.";
    // Вывод информации в окно
    WshShell.Popup(s,0,"Данные о работнике");
    }
  else
    WScript.Echo("Данные на",s,"не найдены");
</script>
</job>
```

6. После выполнения сценария требуемая информация будет записана в файл `listing.txt`:

```
var fso, rf, NameSpaceObj, ForWriting = 2,
    DomObj, E, E1, UserObj;
fso = new ActiveXObject("Scripting.FileSystemObject");
rf = fso.OpenTextFile("listing.txt", ForWriting, true );
NameSpaceObj = GetObject("WinNT:");
NameSpaceObj.Filter = Array("domain");
E=new Enumerator(NameSpaceObj);
List='';
while (!E.atEnd()) {
  DomObj=E.item();
  WScript.Echo("Домен:",DomObj.Name);
  rf.WriteLine("Домен: "+DomObj.Name);
  E1=null;
  E1=new Enumerator(DomObj);
  while (!E1.atEnd()) {
    UserObj=E1.item();
    if (UserObj.Class == "User") {
      WScript.Echo("   "+UserObj.Name);
      rf.WriteLine("   "+UserObj.Name);
      }
    E1.moveNext();
    }
  E.moveNext();
  }
rf.Close();
```

ПРИЛОЖЕНИЕ 5

Ссылки на ресурсы Internet

Журналы и статьи

Адрес	Описание
http://msdn.microsoft.com/msdnmag /default.asp	Журнал "MSDN Magazine". Публикуются статьи по различным технологиям Microsoft
http://msdn.microsoft.com/library /default.asp?url=/library/en-us/dnclinic /html/vbsvjs.asp	"MSDN Online Voices". Здесь находятся статьи разработчика Microsoft Эндрю Клиника (Andrew Clinick), посвященные сценариям WSH
http://www.ddj.com/topics/altlang/	Журнал "Dr. Dobb's Journal", раздел "Scripting and Alternative Languages Discussion Forum"
http://www.win32scripting.com/	Журнал "Windows Scripting Solutions". Освещаются различные аспекты использования сценариев и командных файлов в Windows

Сайты компании Microsoft

Адрес	Описание
http://msdn.microsoft.com/scripting/	Сайт Microsoft Windows Script Technologies, посвященный ActiveX-сценариям. Отсюда можно скачать последнюю версию WSH, документацию по WSH, WSC, JScript, VBScript, отладчики сценариев

(окончание)

Адрес	Описание
http://msdn.microsoft.com /developer/default.htm	Электронная библиотека MSDN (Microsoft Developer Network) содержит подробную информацию об объектах автоматизации, которые может использовать WSH, в том числе об объектах ADO и ADSI
microsoft.public.scripting.wsh	Телеконференция Microsoft, посвященная WSH

Зарубежные сайты

Адрес	Описание
http://communities.msn.com/windowsscript/	Документация, статьи, примеры сценариев, ответы на часто задаваемые вопросы (Frequently Asked Questions, FAQ)
http://scripting.winguides.com/	Документация и примеры сценариев JScript и VBScript
http://www.winscripter.com/	Статьи, примеры сценариев, ссылки на сайты схожей тематики
http://cwashington.netreach.net/	Информация по различным языкам сценариев, включая командные файлы Windows NT
http://www.mundspring.com/~mark_baker/	Часто задаваемые вопросы по WSH и смежным технологиям
http://www.netspace.net.au/~torrboy /code/jscriptfaq/	Часто задаваемые вопросы по языку JScript
http://www.activestate.com	Модули для WSH, поддерживающие языки Active Perl, Active Python, Active XSLT

Российские сайты

Адрес	Описание
http://www.scripting.vlink.ru	Сайт посвящен использованию языков сценариев VBScript и JScript, их расширенному применению с использованием ActiveX-элементов. Здесь помещена книга А. В. Неверова "Windows Scripting Host 2.0 (с использованием MS Visual Basic Script и MS JScript)"
http://scripting.narod.ru	Часто задаваемые вопросы по WSH, избранные статьи из Microsoft Knowledge Base, статьи о сценариях, ссылки на сайты, посвященные сценариям

Список литературы

1. Администрирование сети Microsoft Windows NT 4.0. Учебный курс: Пер. с англ. — М.: Издательский отдел "Русская редакция" ТОО "Channel Trading Ltd", 1998. — 496 с.
2. Поддержка Microsoft Windows NT 4.0. Учебный курс: Пер. с англ. — М.: Издательский отдел "Русская редакция" ТОО "Channel Trading Ltd", 1998. — 680 с.
3. Ресурсы Microsoft Windows NT Server 4.0. Книга 1: Пер. с англ. — СПб.: BHV — Санкт-Петербург, 1998. — 408 с.
4. Айвенс К. Эксплуатация Windows NT. Проблемы и решения: Пер. с англ. — СПб.: BHV — Санкт-Петербург, 1999. — 592 с.
5. Баррет Д. JavaScript Web профессионалам: Пер. с англ. — К.: BHV — Киев, 2001. — 240 с.
6. Вайк А. JavaScript в примерах: Пер. с англ. — М.: ДиаСофт, 2000. — 304 с.
7. Карп Д. Хитрости Windows 98: для профессионалов: Пер. с англ. — СПб.: Питер, 2001. — 416 с.
8. Хиллей В. Секреты Windows NT 4.0: Пер. с англ. — К.: Диалектика, 1997. — 528 с.
9. Экк Т. Сценарии ADSI для системного администрирования Windows NT/2000: Пер. с англ. — М., СПб., К.: Издательский дом "Вильямс", 2000. — 576 с.
10. Корнелл Г. Сценарии Windows для работы с файлами // PC Magazine. 1998. № 9 (http://www.pcmag.ru/archive/9809/099824.asp).
11. Уэллс Б. Регистрационные сценарии WSH // Windows 2000 Magazine/RE. 1999. № 1 (http://www2.osp.ru/win2000/1999/01/55.htm).

12. Уэллс Б. Основы WSH // Windows 2000 Magazine/RE. 1999. № 2 (http://www2.osp.ru/win2000/1999/02/14.htm).

13. Уэллс Б. Extensible Markup Language. Роль языка XML в Windows Scripting Host 2.057 // Windows 2000 Magazine/RE. 2000. № 3 (http://www2.osp.ru/win2000/2000/03/056.htm).

14. Уэллс Б. Файлы Windows Script в действии // Windows 2000 Magazine/RE. 2000. № 4 (http://www2.osp.ru/win2000/2000/04/063.htm).

15. Windows Script Host для входа в систему // Lan/Журнал сетевых решений. 1999. № 11 (http://www.osp.ru/lan/1999/11/005.htm).

16. Born G. Microsoft Windows Script Host 2.0 Developer's Guide. — Microsoft Press. 2000. — 528 p.

17. Esposito D. Windows Script Host Programmer's Reference. — Wrox Press. 1999. — 373 p.

18. Fredel T. SAMS Teach Yourself Windows Scripting Host in 21 days. — SAMS. 1999. — 624 p.

19. Frisch A. Windows 2000 Commands Pocket Reference. — O'Reilly & Associates. 2001. — 126 p.

20. Hill T. Windows NT Shell Scripting. — New Riders Publishing. 1998. — 400 p.

21. Hill T. Windows Script Host. — New Riders Publishing. 1999. — 430 p.

22. Stanek W.R. Windows NT Scripting Administrator's Guide. — Hungry Minds. 1999. — 400 p.

23. Stanek W.R. Windows 2000 Scripting Bible. — Hungry Minds. 2000. — 667 p.

24. Weltner T. Windows Scripting Secrets. — Hungry Minds. 2000. — 751 p.

Предметный указатель

Л

М

О

П

Р

С

Ф

Ш