Информация об авторе:

Автор: Поляков Андрей Валерьевич
Web: http://info-master.su
e-mail: mail@info-master.su
Блог: av-inf.blogspot.ru
В контакте: vk.com/id185471101
Фейсбук: facebook.com/100008480927503

Страница книги: http://av-mag.ru/books/vbscript.htm

Эта книга не закончена. Возможно, более полную версию вы найдёте на странице книги (см. выше).

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых автором как надёжные. Тем не менее, имея в виду возможные человеческие или технические ошибки, автор не может гарантировать абсолютную точность и полноту приводимых сведений и не несёт ответственности за возможные ошибки и ущерб, связанные с использованием этого документа.

1. РАЗРЕШЕНИЯ

Разрешается использование книги в ознакомительных и образовательных целях (только для личного использования).

2. ОГРАНИЧЕНИЯ

Запрещается использование книги в коммерческих целях (продажа, включение в состав платных продуктов и т.п.). Запрещается размещение книги на любых Интернет-ресурсах. Запрещается вносить изменения в текст книги. Запрещается присваивать авторство. Запрещается распространять книгу любыми способами.

Поляков Андрей Валерьевич

VISUAL BASIC SCRIPT ДЛЯ НАЧИНАЮЩИХ И БЫВАЛЫХ

Курган 2011 г.

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Данный документ содержит описание языка сценариев **Visual Basic Scrip (VBScript)**. Здесь речь пойдёт о **VBScript 5.6**, но практически всё, что здесь написано, будет справедливо как для более ранних, так и для более поздних версий. Приведённые здесь статьи основаны на официальной документации по VBScript 5.6, которая доступна для свободного скачивания на сайте Microsoft (документация на английском языке).

Несмотря на то, что в основе этого документа лежит справочный материал по VBScript, этот документ можно считать самостоятельным, поскольку представленный здесь материал – это не просто сухой перевод. Материалы снабжены большим количеством дополнительных пояснений и примеров. Исправлены некоторые ошибки, имеющиеся в оригинальной документации. Если вы скачали этот документ с сайта автора, то вместе с ним вы получили исходные коды всех примеров, рассмотренных в этом документе. Если же нет, то исходные коды и свежую версию этого документа вы можете скачать здесь:

http://www.avprog.narod.ru/progs/vbs/vbs-content.htm

Поскольку книга (будем называть это так) ещё не закончена, то рекомендую вам периодически заглядывать на указанную выше страницу для того, чтобы получить новую версию книги.

1. ОБЩИЕ СВЕДЕНИЯ

Сценарий или скрипт (от английского слова **Script** – сценарий) – это исходный код, который, как правило (но не обязательно), встраивается в какой-либо документ, интернет-страницу и т.п. Все вы видели на страницах Интернета различные кнопки, флажки и прочие элементы управления. При нажатии на такую кнопку обычно выполняется какой-либо сценарий. Скриптовых языков довольно много. Вот лишь некоторые из них:

- JavaScript
- JScript (не путайте с JavaScript)
- Visual Basic Script (VBScript)
- Delphi Script
- Action Script
- PHP

Скрипты, написанные на некоторых языках (например, JScript, VBScript) можно сохранить в виде файла, а затем выполнить двойным щелчком левой кнопки мыши на компьютере под управлением Windows.

Скрипты работают медленнее, чем откомпилированные программы. Но зато у скриптов есть одно неоспоримое преимущество – для их создания и выполнения не требуются ни среда разработки (достаточно простого текстового редактора) ни компилятор (скрипты выполняются программными средствами, входящими в состав операционной системы).

VBScript используется не так широко, как, например, JavaScript. Однако я выбрал именно его. Почему? Потому что:

1. Пожалуй, это самый простой скриптовый язык из всех мне известных
2. Этот язык, поскольку он очень простой, часто используется для разработки в специализированных программах, таких как ПАРУС, SCADA-системы и т.п.
3. Компания Microsoft объявляет о том, что будет в обозримом будущем поддерживать язык VBScript, что в определённой степени даёт уверенность в завтрашнем дне программистам, использующим этот язык

VBScript – это довольно мощное средство. Невозможно одновременно выложить всю информацию в книге. Поэтому данный документ будет постоянно обновляться и «обрастать» новыми материалами. Если вы по каким-либо причинам не хотите переводить с английского оригинальную документацию, то добавьте страницу http://www.avprog.narod.ru/progs/vbs/vbs-content.htm в закладки и время от времени заходите на неё. Тем более что приведённые здесь материалы – это не тупой перевод оригинальной документации, а более подробное описание приёмов работы с VBScript с многочисленными примерами.

VBScript – это продукт компании Microsoft, поэтому нет никаких гарантий, что ваши скрипты будут одинаково хорошо работать со всеми браузерами. Точнее, в настоящее время VBScript поддерживается только Internet Explorer. Если это вас не пугает, тогда начнём…

1.1. Что такое VBScript

Microsoft Visual Basic Scripting Edition предоставляет активные сценарии для различных сред окружения, включая сценарии Web-клиента в Microsoft Internet Explorer и сценарии Web-сервера в Microsoft Internet Information Service.

Лёгкий для использования и изучения

Если вы уже знаете Visual Basic или Visual Basic for Applications (VBA), то VBScript покажется вам очень привычным и знакомым. Даже если вы не знаете Visual Basic, то изучение VBScript поможет вам в дальнейшем освоить программирование на языках, подобных Visual Basic. Хотя, если вы изучите только основы VBScript и создадите несколько простых интернет-страниц, то это не научит вас программированию. Изучение программирования – это отдельная тема. Некоторые вопросы программирования освещены в различных статьях на сайте автора http://www.avprog.narod.ru.

Для изучения данного материала вам понадобятся базовые знания в области программирования (хотя бы на уровне понимания что такое идентификатор, переменная, тип данных и т.п.) и знание основ HTML.

Windows Script

VBScript предназначен для работы в приложениях, использующих Windows Script. С Windows Script браузеры и другие приложения не требуют специальной интеграции кода для каждого компонента сценария. Windows Script включает в себя компилятор сценариев и менеджер пространства имён, доступных для разработчика. Microsoft будет предоставлять поддержку для VBScript. Microsoft непрерывно работает с поставщиком Internet для определения стандарта Windows Script, так что некоторые решения могут быть изменены. Windows Script использует Microsoft® Internet Explorer и Microsoft® Internet Information Service.

VBScript в других приложениях и браузерах

Как разработчик вы можете разрешить выполнение кода VBScript, не загружая его в ваш программный продукт. Microsoft предоставляет бинарное выполнение VBScript для 32-разрядных Windows® API, 16-разрядных Windows API и Macintosh®. VBScript встраивается в браузеры World Wide Web (WWW). VBScript и Windows Script могут также использоваться как основные языки сценариев в других приложениях. Скрипты, написанные на VBScript и встроенные в HTML-страницу гарантированно будут работать только при использовании Internet Explorer. Вопросам обеспечения работоспособности скриптов в других браузерах будет посвящено несколько разделов далее.

1.2. Как начать?

Обычно изучение нового языка программирования начинают с какой-либо простой программы. По традиции, эта программа выводит на экран строку «Hello, WORLD!». Мы последуем этой традиции и создадим скрипт, который будет выводить на экран диалоговое окно с надписью «Hello, WORLD!».

Поскольку встраивать коды VBScript в HTML-страницу мы пока не умеем (об этом в следующих разделах), то сохраним наш скрипт в виде файла. Для этого в любом простом текстовом редакторе (например, в Блокноте) создадим файл и сохраним его с расширением vbs. Присвоим ему имя, например, HELLO.VBS. Создавать файлы со сценариями можно и в мощных текстовых редакторах, например, в MS Word, однако учтите, что это должен быть простой текстовый файл, иначе скрипт работать не будет. Поэтому, чтобы избежать неприятностей, лучше воспользоваться простым текстовым редактором, таким как Блокнот.

В то же время Блокнот не является лучшим решением, так как текст на экране будет одного цвета, что не очень удобно, особенно при создании сложных сценариев. Поэтому разработчики сценариев обычно используют специальные текстовые редакторы с подсветкой синтаксиса. В таких редакторах ключевые слова, комментарии и прочая служебная информация выделяется цветом, поэтому читать такой текст очень легко и удобно. Могу порекомендовать абсолютно бесплатный, но в то же время очень продвинутый текстовый редактор PSPad, скачать который можно здесь:

http://www.pspad.com/ru/

Но ближе к делу. Итак, мы создали файл HELLO.VBS. Если вы всё сделали правильно, то значок файла в Проводнике должен выглядеть так:

Откроем этот файл с помощью текстового редактора и запишем там всего одну строку:

```
MsgBox "Hello, WORLD!"
```

Сохраним файл. А теперь попробуйте открыть его обычным для Windows способом, то есть двойным щелчком левой кнопки мыши. Если вы не допустили ошибок в тексте (текст настолько простой, что сделать это почти нереально, но теоретически возможно))), то сценарий будет выполнен и на экране появится окно с надписью «Hello, WORLD!» и кнопкой OK (см. рис. 1).

Рис. 1. Результат выполнения сценария.

Ну вот и всё. Ваш первый сценарий готов. Разбирать здесь особо нечего. Всем, надеюсь, понятно, что функция **MsgBox** выводит на экран строку, которая передаётся в функцию в качестве параметра. Строка выводится в диалоговом окне с кнопкой OK. Впрочем, эта функция не так проста, как вы, быть может, подумали. Но о различных способах обмена данными с пользователем мы поговорим в следующий раз…

ПРИМЕЧАНИЕ
Если у вас всё-таки не получилось создать свой первый сценарий, то найдите папку SOURCE, которую вы скачали вместе с этим документом, а в ней папку CH_01, в которой и будет исходный файл этого сценария **hello.vbs**.

1.3. Добавление кода VBScript в HTML-страницу

Вы можете использовать элемент SCRIPT для добавления кода VBScript в HTML-страницу.

1.3.1. Тег <SCRIPT>

Исходный код сценария VBScript записывается между парными тегами <SCRIPT>. Например, следующая процедура проверяет дату:

```
<SCRIPT LANGUAGE="VBScript">
<!--
   Function CanDeliver(Dt)
      CanDeliver = (CDate(Dt) - Now()) > 2
   End Function
-->
</SCRIPT>
```

Начальный и конечный теги <SCRIPT> заключают в себя исходный код сценария. Атрибут LANGUAGE указывает, какой язык будет использоваться в этом сценарии. Вы обязательно должны указывать язык, потому что браузеры могут использовать и другие языки. Обратите внимание, что функция **CanDeliver** заключена в теги комментариев (<!-- и -->). Это указывает браузеру, что исходный код, заключённый в теги <SCRIPT>, не нужно отображать на экране.

Приведённый выше пример – это основная функция, которая не связана с какими-либо элементами управления, вы можете включить её в раздел HEAD страницы HTML:

```
<HTML>
<HEAD>
<TITLE>Place Your Order</TITLE>
<SCRIPT LANGUAGE="VBScript">
<!--
   Function CanDeliver(Dt)
      CanDeliver = (CDate(Dt) - Now()) > 2
   End Function
-->
</SCRIPT>
</HEAD>
<BODY>
...
```

Вы можете использовать блок SCRIPT в любом месте HTML-страницы. Вы можете разместить этот блок как в секции BODY, так и в секции HEAD. Однако вы, вероятно, захотите разместить все сценарии в секции HEAD, чтобы хранить все сценарии в одном месте. Хранение вашего кода в секции HEAD гарантирует, что весь код будет прочитан и интерпретирован перед вызовом функций из секции BODY.

Одно важное исключение из этих правил – когда вы захотите передать код сценария между формами для связи с событиями объектов в вашей форме. Например, вы можете вставить исходный код сценария, который выполняется при щелчке на кнопке, которая находится на вашей форме:

```
<HTML>
<HEAD>
<TITLE>Test Button Events</TITLE>
</HEAD>
<BODY>
<FORM NAME="Form1">
   <INPUT TYPE="Button" NAME="Button1" VALUE="Click">
   <SCRIPT FOR="Button1" EVENT="onClick" LANGUAGE="VBScript">
      MsgBox "Button Pressed!"
   </SCRIPT>
</FORM>
</BODY>
</HTML>
```

Большая часть вашего кода будет определена в подпрограммах **Sub** или **Function**, которые будут вызываться при необходимости. Однако вы можете создавать с помощью VBScript внешние процедуры, но объявлять их в блоке SCRIPT. Этот код выполняется только один раз при загрузке страницы. Это позволяет вам инициализировать данные или динамически изменять вид вашей страницы при загрузке. Более подробно применение сценариев на веб-страницах рассмотрено в следующих разделах.

2. УРОКИ VBSCRIPT

2.1. Основы VBScript

2.1.1. Типы данных

Для работы с любым языком программирования в первую очередь необходимо хотя бы в общих чертах изучить его синтаксис и типы данных. В этом разделе коротко расскажем о типах данных, которые используются в VBSCript.

Вообще-то VBScript имеет только один тип данных, который называется **Variant**. Тип Variant – это специальный тип данных, который может содержать данные различных типов. То есть вам нет необходимости объявлять переменную конкретного типа. Достаточно просто объявить имя переменной, а какие данные будут в этой переменной - интерпретатор определит автоматически во время выполнения скрипта.

Тип Variant может содержать как числовые, так и строковые данные. Данные интерпретируются как числовые, когда в вашей программе вы используете их в математических выражениях и как текстовые, когда вы используете их при работе со строками. Таким образом, если VBScript распознаёт ваши данные как числовые, то с этими данными можно работать как с числами. Если же данные содержат символы, отличные от применяемых в числах, то эти данные интерпретируются как строковые. Если вы хотите, чтобы числовые данные интерпретировались VBScript как строковые, то вы должны заключать их в двойные кавычки (" ").

Например, мы объявили переменную **Num**. Тогда в сценарии мы можем использовать её следующим образом:

```
Num = 5 + 10
Num = "Результат: " & Num
MsgBox Num
```

Этот сценарий выполнится без ошибки. В первой строке мы используем переменную **Num** как целое число. Интерпретатор это определяет по операнду "+", то есть операция "5 + 10" - это операция сложения чисел, результатом которой будет число 15. Следующая строка - это операция сложения строк, интерпретатор определяет это по операнду "&", который используется при сложении строк. Результатом этой операции уже будет НЕ ЧИСЛО, а СТРОКА "15", которую мы и выводим на экран с помощью функции **MsgBox**. А вот если бы в этом сценарии вместо

```
Num = "Результат: " & Num
```

мы бы написали

```
Num = "Результат: " + Num
```

то сценарий не был бы выполнен и вы бы получили сообщение об ошибке, потому что нельзя с помощью математических операндов складывать строки и числа (хотя можно складывать две строки!!!, об этом мы ещё поговорим в следующих разделах).

Так как VBScript имеет только один тип данных Variant, то и все функции VBScript также всегда возвращают данные типа Variant.

Variant: подтипы

Кроме простых числовых и строковых типов данных Variant включает в себя различные представления числовой информации. Например, вы можете оперировать числовой информацией, которая представляет дату и время. Если используются данные типа даты и времени, то результат всегда представляется как дата или время. Вы можете также использовать различную числовую информацию в широком диапазоне значений от типа Boolean до чисел с плавающей точкой. Эти категории данных, которые могут содержаться в типе Variant называются подтипами. Однако учтите, что вы можете работать с этими данными, только используя тип Variant, и тип Variant должен вести себя в соответствии с подтипом данных, который в нем содержится.

Впрочем, VBScript позволяет очень вольно обращаться с данными. Например, такой код будет выполнен без сообщений об ошибках:

```
Num = True
Num = Num + 10
```

Хотя на первый взгляд может показаться, что логические переменные нельзя складывать с целочисленными. Оказывается, что в VBScript это вполне допустимо, вот только результат может оказаться для вас неожиданным. Поэтому, несмотря на все преимущества использования типа Variant, делать это нужно с умом, понимая возможные последствия автоматического преобразования типов.

В следующей таблице приведены типы данных, которые могут содержаться в типе Variant.

Подтип	**Описание**
Empty	Инициализация типа Variant. Содержит 0 для числовых значений и пустую строку (“”) для строковых значений.
Null	Не содержит никаких значащих данных.
Boolean	Логический тип, может принимать значения True (истина) или False (ложь)
Byte	Целое число в диапазоне 0…255.
Integer	Целое число в диапазоне -32768…32767
Currency	Денежный тип, диапазон от -922337203685477,5808 до 922337203685477,5807.
Long	Целое число в диапазоне -2147483648…2147483647
Single	Число с плавающей точкой в диапазоне от -3.402823E38 до -1.401298E-45 для отрицательных значений и от 1.401298E-45 до 3.402823E38 для положительных значений.
Double	Число с плавающей точкой в диапазоне от -1.79769313486232E308 до -4.94065645841247E-324 для отрицательных значений и от 4.94065645841247E-324 до 1.79769313486232E308 для положительных значений.
Date (Time)	Числовое значение, которое представляет дату в диапазоне от 1-го января 100 года до 31 декабря 9999 года.
String	Строка переменной длины, которая может быть длиной до 2 миллиардов символов.
Object	Объект.
Error	Код ошибки.

Вы можете использовать стандартные функции преобразования типов для преобразования данных из одного подтипа в другой. Кроме того, функция **VarType** возвращает информацию о том, какие данные хранятся в вашей переменной типа Variant.

2.1.2. Переменные VBScript

Что такое переменная

Переменная — это ссылка на местоположение в памяти компьютера, где вы можете сохранять программную информацию, которая может изменяться в процессе выполнения сценария. Переменная имеет имя (идентификатор). Таким образом, вместо обращения к памяти по адресу, вы можете просто объявить переменную, присвоив ей какое-либо имя и далее обращаться к ней по имени. Например, вы можете создать переменную с именем **ClickCount** и сохранять в ней количество щелчков пользователя по отдельной веб-странице. Вам нет необходимости знать, где именно в памяти компьютера хранится это значение. Вам достаточно помнить имя переменной, которое используется в качестве ссылки на область памяти. Используя это имя вы можете получить или изменить хранящееся в памяти значение. В **VBScript** все переменный имеют тип **Variant** (см. раздел «Типы данных VBScript»).

Объявление переменных

Для явного объявления переменной в вашем скрипте используйте операторы **Dim**, **Public** или **Private**. Например:

```
Dim DegreesFahrenheit
```

Чтобы объявить несколько переменных, разделяйте их запятой, например:

```
Dim Top, Bottom, Left, Right
```

Несколько переменных можно также объявить в нескольких строках:

```
Dim Top       'Верхняя координата окна
Dim Bottom    'Нижняя координата окна
Dim Left      'Левая координата окна
Dim Right     'Правая координата окна
```

Этот вариант более удобен в тех случаях, когда требуется разместить комментарий для каждой переменной.

Вы можете также объявить переменную неявно, используя её имя в любом месте вашего сценария. Неявное объявление — это когда вы предварительно не объявляете переменную, а просто вводите имя переменной в любом месте сценария и используете её. В общем случае это не рекомендуется, так как это может привести к потенциальным ошибкам в сценарии. Например, вы явно объявили переменную **Top**, а затем забыли про это и объявили переменную **Top** неявно. То есть получится, что в разных местах сценария вы по разному будете использовать одну и ту же переменную, что может привести к непредсказуемым последствиям выполнения сценария. Чтобы этого избежать, рекомендуется использовать оператор **Option Explicit**, который требует явного объявления всех переменных. Оператор **Option Explicit** должен быть первым оператором в вашем сценарии. Два примера:

```
s = "Test VAR"
MsgBox s
```

Этот сценарий выполнится без ошибок, хотя переменная **s** не объявлена явно.

```
Option Explicit

s = "Test VAR"
MsgBox s
```

При попытке выполнить этот сценарий произойдёт ошибка, так как переменная **s** не объявлена (см. рис. 2.1).

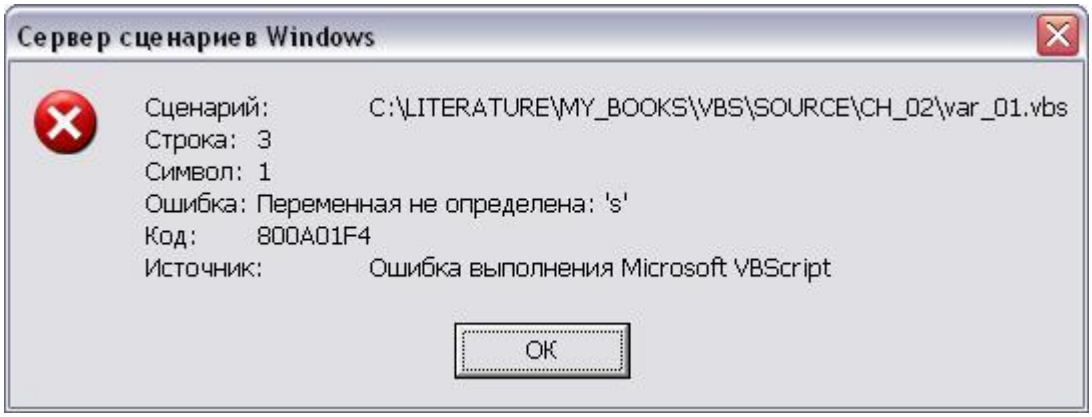

Рис. 2.1. Ошибка сценария.

Таким образом, если вы не хотите случайно использовать неявно объявленные переменные, то используйте оператор **Option Explicit**. Случайно можно объявить неявно переменную, например, из-за опечатки. Допустим, у вас есть явно объявленная переменная Right, а где-то в тексте сценария вы случайно напечатали вместо слова Right слово Riht. Если вы не использовали оператор **Option Explicit**, то это не будет считаться ошибкой, так как Riht будет расцениваться как неявно объявленная переменная. Однако результат выполнения такого сценария будет непредсказуем, так как написав Riht, вы подразумевали Right, но не заметили этой опечатки и свято верите в то, что присвоили значение переменной Right, хотя на самом деле это была переменная Riht.

Правила именования переменных

При именовании переменных следует выполнять стандартные правила именования всех идентификаторов в **VBScript**. Имя переменной:

- Должно начинаться с буквы
- Не должно содержать даты и времени
- Не должно превышать 255 символов
- Должно быть уникальным в пределах области, где оно объявлено

Область видимости и время жизни переменных

Область видимости — это границы, в которых объявленную в этих границах переменной можно использовать. Например, переменную, явно объявленную в процедуре, можно использовать только в этой процедуре. Такая переменная является локальной и называется переменной уровня процедуры. Если вы объявляете переменную вне процедуры, то такая переменная видна из любого места сценария. Это переменная является глобальной и называется переменной уровня сценария.

Время жизни переменной — это время, в течение которого существует переменная. Время жизни переменной уровня сценария начинается с момента её объявления и длится до окончания выполнения сценария. Время жизни переменной уровня процедуры начинается с момента вызова процедуры и продолжается до окончания выполнения процедуры. При выходе из процедуры локальная переменная уничтожается. Использование локальных переменных позволяет экономить память компьютера. Локальные переменные могут иметь такие же имена, как глобальные. В этом случае в процедуре используется локальная переменная, а в других местах сценария используется глобальная переменная.

Присваивание значений переменным

Для присваивания значения переменной используется выражение следующего вида (слева находится имя переменной, справа — присваиваемое значение):

```
B = 200
```

После выполнения этого кода переменной B будет присвоено значение 200. В качестве операнда присваивания используется символ «=».

Скалярные переменные и массивы

Наиболее часто вам приходится присваивать значения одиночным переменным. Переменная, содержащая одно значение — это **скалярная переменная**. Но также часто требуется записать в одну переменную несколько значений. Для такого случая вы можете создать переменную, содержащую последовательность значений. Такая переменная называется **массив**. Массивы и скалярные переменные объявляются одинаково, за исключением того, что при объявлении массива после имени переменной используются круглые скобки. В следующем примере объявлен массив A, содержащий 11 элементов:

```
Dim A(10)
```

Здесь в круглых скобках находится число 10, хотя, как уже говорилось, объявлен массив из 11 элементов. Это объясняется тем, что нумерация всех массивов в VBScript начинается с 0, а не с 1. То есть при объявлении массива в круглых скобках вам требуется всегда указывать количество элементов минус 1. Объявленный таким образом массив называется **массив с фиксированным количеством элементов**.

Значение каждому элемента массива присваивается с использованием индекса элемента массива. В нашем примере каждому (из 11) элементу массива от 0 до 10 может быть присвоено значение следующим образом:

```
A(0) = 256
A(1) = 324
A(2) = 100
. . .
A(10) = 55
```

Подобным образом можно получить данные из массива:

```
. . .
SomeVariable = A(8)
. . .
```

Массивы не ограничиваются одним измерением. Вы можете использовать в массивах до 60 измерений, несмотря на то, что люди не могут воспринимать более 3 или 4 измерений. При объявлении массива с несколькими измерениями количество элементов для каждого измерения отделяется запятой. В следующем примере объявлен массив, который представляет собой таблицу значений из 6 строк и 11 столбцов:

```
Dim MyTable(5, 10)
```

В двумерном массиве первое число — это количество строк, второе — количество столбцов.

Вы можете также объявлять массивы, размер которых может изменяться во время выполнения сценария. Такой массив называется **динамический массив**. Динамический массив объявляется с использованием любого из операторов **Dim** или **ReDim**. Однако для динамического массива не указывается количество измерений и элементов в круглых скобках. Например:

```
Dim MyArray()
ReDim AnotherArray()
```

В последствии для использования динамического массива вы должны использовать оператор ReDim для определения количества измерений и элементов в каждом измерении. В следующем примере вначале устанавливается размер массива, равный 25. Затем размер массива изменяется на 30, но используется ключевое слово Preserve, чтобы сохранить уже имеющиеся в массиве данные. Если это слово не использовать, то все имеющиеся в массиве данные при изменении размера будут утеряны.

```
ReDim MyArray(25)
. . .
ReDim Preserve MyArray(30)
```

Вы можете изменять размер массива неограниченное количество раз, однако вы должны знать, что при уменьшении размера массива вы потеряете данные, которые находятся в исключённых элементах.

2.1.3. Константы VBScript

Что такое константа

Константа — это понятное имя, которое используется вместо числового или строкового значения. Значение константы, в отличие от значения переменной, никогда не изменяется. VBScript имеет некоторое количество предопределённых констант, которые могут быть переопределены пользователем. Константы используются в тех случаях, когда какое-то значение часто используется в сценарии. Например, в вашем сценарии часто выводится сообщение об окончании какого-либо процесса. Тогда вы можете константу и присвоить ей это сообщение. Если потом вы решите изменить это сообщение, то вам не придётся изменять его во всём сценарии, достаточно будет только изменить это сообщение при объявлении константы.

Создание констант

Определённые пользователем константы в VBScript создаются с помощью оператора **Const**. Используя этот оператор, вы можете создавать строковые или числовые константы с понятными именами и присваивать им значения. Например:

```
Const MyString = "Это моя строка."
Const MyAge = 49
```

Учтите, что строка символов должна заключаться в двойные кавычки (" "). Очевидно, что двойные кавычки используются для того, чтобы отличать строковые значения от числовых. При использовании даты и времени требуется заключать их в знаки числа (#). Например:

```
Const CutoffDate = #6-1-97#
```

Вы можете принять схему именования констант, отличную от схемы именования переменных. Это позволит предотвратить попытку присвоить значение константе в сценарии. Например, вы можете использовать префиксы **vb** для переменных и **con** для констант, или использовать для именования констант символы в верхнем регистре. Различие в именовании переменных и констант предотвратит различные неприятности при разработке комплексных сценариев, состоящих из нескольких файлов.

2.1.4. Операторы VBScript

VBScript имеет полный набор операторов, включающий в себя арифметические операторы, операторы сравнения, операторы конкатенации (объединения) и логические операторы.

Приоритет операторов

Если в одном выражении используются несколько операторов, то каждая часть выражения оценивается и вычисляется в предопределённом порядке в соответствии с приоритетом операторов. То есть очерёдность выполнения операторов зависит от их приоритета. Вы можете использовать круглые скобки для переопределения приоритета и форсировать выполнение какой-либо части выражения, то есть сделать так, что бы эта часть выражения выполнялась первой. Операторы, вложенные в круглые скобки, всегда выполняются раньше, чем операторы, находящиеся вне скобок. Однако операторы в пределах круглых скобок сохраняют стандартные приоритеты.

Если выражение содержит операторы из более чем одной категории, то в первую очередь выполняются арифметические операторы, затем выполняются операторы сравнения, а логические операторы выполняются в последнюю очередь. Все операторы сравнения имеют равный приоритет. Если в выражении имеется несколько операторов сравнения, то они выполняются по очереди слева направо в то порядке, в котором они записаны в выражении. Арифметические и логические операторы вычисляются в порядке, в котором они перечислены в приведённых ниже таблицах.

Если в одном выражении имеются операторы умножения и деления, то эти операторы вычисляются в том порядке, в котором они записаны в выражении, то есть слева направо. Аналогично, если в одном выражении встречаются сложение и вычитание, то то эти операторы вычисляются в том порядке, в котором они записаны в выражении, то есть слева направо.

Арифметические операторы	
Описание	**Символ**
Возведение в степень	^
Унарный минус	-
Умножение	*
Деление	/
Целочисленное деление	\
Модуль	Mod
Сложение	+
Вычитание	-
Конкатенация (соединение) строк	&

Операторы сравнения	
Описание	**Символ**
Равно	=
Не равно	<>
Меньше чем	<
Больше чем	>
Меньше или равно	<=
Больше или равно	>=
Эквивалентность объектов	Is

Логические операторы	
Описание	**Символ**
Логическое НЕ	Not
Логическое сложение	And
Логическое умножение	Or
Исключающее ИЛИ	Xor
Эквивалентность	Eqv
Логическая импликация	Imp

Оператор конкатенации (соединения) строк (&) не является арифметическим оператором, но его приоритет лежит после всех арифметических операторов и перед операторами сравнения. Оператор **Is** — это оператор сопоставления ссылок на объекты. Он не сравнивает объекты и их значения, он только проверяет, не ссылаются ли две переменные на один и тот же объект.

2.1.5. Использование условных операторов

Управление выполнением программы

Вы можете управлять ходом выполнения вашего сценария с помощью циклов и условных операторов. Используя условные операторы вы можете написать код VBScript, который принимает решения и повторяет действия. В VBScript доступны следующие операторы сравнения:

- Оператор **If...Then...Else**
- Оператор **Select Case**

Ветвление программы с использованием If...Then...Else

Оператор **If...Then...Else** используется для проверки условий на соответствие значениям **True** (Истина) или **False** (Ложь). Условие является результатом вычисления какого-либо выражения. Например, результатом выражения (5 + 5 = 10) будет True, так как это утверждение верно. Результатом выражения (2 * 2 = 5) будет False, так как 2 * 2 = 4, а не 5. Таким образом в коде сценария в зависимости от истинности выполнения какого-либо условия мы можем выполнить тот или иной участок кода. Это действие называется ветвлением. То есть сценарий будет выполнять один из двух участков кода, в зависимости от истинности условия, но оба участка кода не могут быть выполнены за один проход сценария. Например:

```
Dim x, y

x = 2
y = 5

If (x * x) = y Then
  MsgBox "Ответ правильный"
Else
  MsgBox "Ответ НЕ правильный"
End If
```

В этом примере, если **y = 5**, то условие **(x * x) = y** НЕ выполняется, то есть равно **False**. Следовательно, будет выполнен код, который находится после слова **Else**. Если вы присвоите переменной **y** значение **4**, то условие **(x * x) = y** будет выполнено, следовательно, в сценарии будет выполнен код, который находится после слова **Then**, а код после слова **Else** выполняться НЕ будет. Вообще. Если вы знаете английский, то вы уже успели заметить, что языковая конструкция **If...Then...Else** переводится как **Если...То...Иначе**. Иными словами, если попытаться перевести этот код на русский язык, то получим примерно следующее:

```
Если ВЫПОЛНЯЕТСЯ ЭТО УСЛОВИЕ То
    Надо выполнить этот участок кода
Иначе (Если условие не выполняется)
    Надо выполнить вот этот участок кода
Конец Если (Конец блока If...Then...Else)
```

Обычно условие — это выражение, в котором используются операции сравнения одного значения с другим, или одной переменной с другой, или переменной и константы, или двух выражений и т.п. Подробности см. в разделе «Операторы сравнения». Оператор **If...Then...Else** может быть многократно вложен в другие операторы сравнения. Таким образом можно получить несколько уровней вложенности условных операторов.

Выполнение операторов, если условие равно TRUE

Для того, чтобы выполнить один оператор, если проверяемое условие равно True, можно использовать однострочный синтаксис оператора **If...Then...Else**. В следующем примере показан однострочный синтаксис. Учтите, что этот пример не включает ключевое слово **Else**.

```
Sub FixDate()
  Dim myDate
  myDate = #2/13/95#
  If myDate < Now Then myDate = Now
End Sub
```

Для того чтобы выполнить несколько строк кода, вы должны использовать многострочный синтаксис (блок кода). Этот синтаксис содержит оператор **End If**, как показано в следующем примере:

```
Sub AlertUser(value)
  If value = 0 Then
    AlertLabel.ForeColor = vbRed
    AlertLabel.Font.Bold = True
    AlertLabel.Font.Italic = True
  End If
End Sub
```

Выполнение одного блока операторов, если условие равно TRUE, и другого блока операторов, если условие равно FALSE

Вы можете использовать оператор **If...Then...Else** для определения двух блоков выполняемых операторов (это уже рассматривалось выше, но для закрепления материала приведём ещё один пример). Один блок выполняется, если условие выполняется (равно TRUE), другой блок выполняется, если условие НЕ выполняется (равно FALSE).

```
Sub AlertUser(value)
  If value = 0 Then
    AlertLabel.ForeColor = vbRed
    AlertLabel.Font.Bold = True
    AlertLabel.Font.Italic = True
  Else
    AlertLabel.Forecolor = vbBlack
    AlertLabel.Font.Bold = False
    AlertLabel.Font.Italic = False
  End If
End Sub
```

Выбор между несколькими альтернативами

Вариации на тему **If...Then...Else** позволяют вам выбирать из нескольких альтернатив. Добавление ключевого слова **ElseIf** расширяет функциональность оператора **If...Then...Else** таким образом, что вы можете управлять потоком программы не по двум, а по нескольким направлениям. Например:

```
Sub ReportValue(value)
  If value = 0 Then
    MsgBox value
  ElseIf value = 1 Then
    MsgBox value
  ElseIf value = 2 then
    Msgbox value
  Else
    Msgbox "Значение вне диапазона!"
  End If
End Sub
```

Вы можете добавлять столько раз слово **ElseIf**, сколько это необходимо, и таким образом предоставлять альтернативный выбор. Многократное использование **ElseIf** часто приводит к громоздким и плохо читаемым участкам кода. Поэтому в таких случаях лучше использовать альтернативный путь — оператор **Select Case**.

Использование оператора Select Case

Конструкция **Select Case** является альтернативой оператору **If...Then...ElseIf** для выборочного выполнения блока операторов из множества имеющихся вариантов кода. Оператор **Select Case** по сути похож на оператор **If...Then...ElseIf**, но делает код более эффективным и читаемым.

Конструкция **Select Case** работает с одним выражением, которое вычисляется один раз (см. первую строчку в приведённом ниже примере). Результат вычисления этого выражения сравнивается в каждом операторе **Case** конструкции **Select Case**. Если найдено значение, равное результату вычисления выражения, то выполняется блок операторов, который следует за словом **Case** до того места, пока не встретится слово **Case** или **End Select**. Пример:

```
Select Case x
  Case x = 1
    MsgBox "x = 1"
  Case x = 2
    MsgBox "x = 2"
  Case x = 3
    MsgBox "x = 3"
  Case Else
    MsgBox "Не удалось определить X"
End Select
```

В этом примере проверяется значение переменной **x**. Если оно равно 1, то на экран выводится «x = 1», если оно равно 2, то на экран выводится «x = 2», если оно равно 3, то на экран выводится «x = 3», если **x** равно другому значению, то на экран выводится «Не удалось определить X». То есть если среди представленных значений не было обнаружено равное **x**, то выполняются операторы, которые следуют за словом **Case Else**. Этот оператор можно опустить, тогда в случае несоответствия вычисленному значению ни одного из представленных не будет выполнен ни один блок конструкции **Select Case**.

Учтите, что в конструкции **Select Case** значение выражения вычисляется только один раз, в отличие от оператора **If...Then...ElseIf**, в котором выполняется вычисление результата сравнения в каждом операторе **ElseIf**. Вы можете использовать конструкцию **Select Case** вместо оператора **If...Then...ElseIf** только в том случае, если все выражения оператора **ElseIf** аналогичны.

2.1.6. Использование циклов

Использование циклов для повторения кода

Циклы позволяют вам выполнять блок операторов повторно. Некоторые циклы повторяют выполнение операторов, пока условие ложно (равно False), другие повторяют выполнение, пока условие истинно (равно True). Также имеются циклы, которые выполняются заданное количество раз. В VBScript доступны следующие циклы:

- **Do...Loop**. Цикл выполняется пока условие истинно, или до тех пор, пока условие не станет истинно
- **While...Wend**. Цикл выполняется пока условие истинно.
- **For...Next**. Цикл выполняется заданное количество раз.
- **For Each...Next**. Повторяет выполнение блока операторов для каждого элемента коллекции или каждого элемента массива.

Использование цикла Do Loop

Вы можете использовать конструкцию **Do...Loop** для выполнения блока операторов бесконечное количество раз. Блок операторов выполняется пока условие истинно (равно TRUE) или до тех пор, пока условие не станет истинным.

Выполнение цикла, пока условие истинно

Используйте ключевое слово **While** для проверки условия выполнения цикла **Do...Loop**. Вы можете проверять условие перед выполнением цикла (как показано в примере ChkFirstWhile) или после выполнения цикла (как показано в примере ChkLastWhile). В процедуре ChkFirstWhile переменной myNum будет присвоено значение 10 вместо 20 после выполнения цикла, цикл будет выполнен 10 раз. Если бы переменной myNum перед циклом было присвоено значение 10 или меньше, то цикл не выполнился бы ни разу, потому что условие проверяется перед циклом. В процедуре ChkLastWhile цикл будет выполнен только один раз. И хотя условие выполнения цикла не выполняется, цикл всё-таки будет выполнен один раз, так как условие проверяется в конце цикла.

```
Sub ChkFirstWhile()
  Dim counter, myNum
  counter = 0
  myNum = 20
  Do While myNum > 10
    myNum = myNum - 1
    counter = counter + 1
  Loop
    MsgBox "Цикл выполнен " & counter & " раз, myNum = " & myNum
End Sub
```

```
Sub ChkLastWhile()
  Dim counter, myNum
  counter = 0
  myNum = 9
  Do
    myNum = myNum - 1
    counter = counter + 1
  Loop While myNum > 10
    MsgBox "Цикл выполнен " & counter & " раз, myNum = " & myNum
End Sub
```

Выполнение цикла до тех пор, пока условие не станет истинно

Используйте ключевое слово **Until** для проверки условия выполнения цикла **Do...Loop**. Вы можете проверять условие перед выполнением цикла (как показано в примере ChkFirstUntil) или после выполнения цикла (как показано в примере ChkLastUntil). Цикл выполняется, пока условие ложно (то есть пока myNum НЕ РАВНО 10). Как только myNum будет равно 10, выполнение цикла прекратится.

```
Sub ChkFirstUntil()
  Dim counter, myNum
  counter = 0
  myNum = 20
  Do Until myNum = 10
    myNum = myNum - 1
    counter = counter + 1
  Loop
    MsgBox "Цикл выполнен " & counter & " раз, myNum = " & myNum
End Sub

Sub ChkLastUntil()
  Dim counter, myNum
  counter = 0
  myNum = 1
  Do
    myNum = myNum + 1
    counter = counter + 1
  Loop Until myNum = 10
    MsgBox "Цикл выполнен " & counter & " раз, myNum = " & myNum
End Sub
```

Прерывание цикла

Вы можете прервать цикл, не дожидаясь выполнения условия завершения цикла. Для принудительного выхода из цикла используется оператор **Exit Do**. Поскольку обычно прерывание цикла требуется в особых ситуациях, например, для того, чтобы избежать попадания в бесконечный цикл, то вы должны использовать оператор **Exit Do** в блоке операторов конструкции **If...Then...Else**. Например, если какое-то условие способствует попаданию в бесконечный цикл, то при его возникновении можно выполнить оператор **Exit Do**, иначе цикл продолжит работу в обычном режиме.

```
Sub ExitExample()
  Dim counter, myNum
  counter = 0
  myNum = 9
  Do Until myNum = 10
    myNum = myNum - 1
    counter = counter + 1
    if counter > 100 Then Exit Do
  Loop
    MsgBox "Цикл выполнен " & counter & " раз, myNum = " & myNum
End Sub
```

В представленном выше примере переменной myNum присваивается значение, которое приводит к зацикливанию процедуры, то есть создаёт бесконечный цикл. Если не принять специальных мер, то этот цикл никогда не завершится. Изначально значение переменной myNum равно 9, а в цикле оно уменьшается, то есть это значение никогда не будет равно 10, а значит, условие цикла никогда не выполнится. Чтобы избежать этой ситуации, мы в каждой итерации цикла проверяем значение счётчика (переменной counter). Если цикл выполнился более 100 раз, то мы его принудительно завершаем с помощью оператора **Exit Do**.

Использование цикла While Wend

Оператор **While...Wend** предоставляется в VBScript для тех, кто хорошо знаком с его использованием. Однако, поскольку оператор **While...Wend** является недостаточно гибким, мы рекомендуем использовать вместо него цикл **Do...Loop**.

Использование цикла For Next

Вы можете использовать оператор **For...Next** для организации цикла, который требуется выполнить заданное количество раз. Для организации цикла используйте переменную-счётчик, которая будет автоматически уменьшаться или увеличиваться в каждой итерации цикла.

В следующем примере организован цикл, который вызывает процедуру 50 раз. Оператор **For** определяет переменную-счётчик **x**, для которой указывается начальное и конечное значения. Оператор **Next** увеличивает значение счётчика на 1.

```
Sub DoMyProc50Times()
  Dim x
  For x = 1 To 50
    MyProc
  Next
End Sub
```

Используя ключевое слово **Step**, вы можете увеличивать или уменьшать счётчик на указанное значение (а не на 1, как в предыдущем примере). В следующем примере переменная-счётчик **j** увеличивается на 2 в каждой итерации цикла. Когда цикл завершится, то результат будет равен сумме чисел 2, 4, 6, 8 и 10, то есть 30.

```
Sub TwosTotal()
  Dim j, s, total
  For j = 2 To 10 Step 2
    If j = 10 Then
      s = s & j & ") = "
    Else
      s = s & j & "+"
    End If
    total = total + j
  Next
  MsgBox "Общая сумма: (" & s & total
End Sub
```

Для уменьшения переменной-счётчика используйте отрицательные значения в операторе **Step**. В этом случае вы должны указать конечное значение меньше, чем начальное значение. В следующем примере переменная-счётчик **myNum** уменьшается на 2 в каждой итерации цикла. Когда цикл завершится, то результат будет равен сумме чисел 16, 14, 12, 10, 8, 6, 4 и 2, то есть 72.

```
Sub NewTotal()
  Dim myNum, total
  For myNum = 16 To 2 Step -2
    total = total + myNum
  Next
  MsgBox "Общая сумма: " & total
End Sub
```

Вы можете выйти из цикла до того, как значение переменной-счётчика достигнет конечного. Для этого используется оператор **Exit For**. Рекомендации по принудительному завершению цикла см. выше в примерах для цикла **Do...Loop**.

Использование цикла For Each Next

Цикл **For Each...Next** похож на цикл **For...Next**, только вместо выполнения операторов заданное количество раз, цикл повторяет выполнение блока операторов для каждого элемента коллекции или для каждого элемента массива. Это особенно полезно в тех случаях, когда вы не знаете, сколько элементов содержится в коллекции.

Пример HTML-кода, приведённый ниже, содержит объект **Dictionary**, используемый для размещения текста в нескольких текстовых полях. Этот пример несколько отличается от примера, приведённого в оригинальной документации. В оригинальной документации используется только цикл **For Each...Next**, где сразу выводятся элементы списка **d** в поля ввода/вывода страницы. Такое решение, конечно, делает код более простым, однако на моём компьютере это почему-то не работает (хотя, по идее, должно работать). Если интересно, можете найти этот пример в оригинальной документации и сравнить. Здесь же приводится доработанный мной пример, который работает на моём компьютере и, скорее всего, будет работать и на вашем.

В оригинальном примере в цикле **For Each...Next** почему-то не выполняется код

```
Document.frmForm.Elements(i).Value = d.Item(i)
```

то есть именно вывод на HTML-страницу. Остальные операторы работают нормально. С чем это связано сказать трудно. Возможно, что это «глюк» используемого мною браузера, а может быть и другая причина.

А теперь коротко рассмотрим пример. Сначала мы создаём список (словарь — объект **Dictionary**), в который заносим несколько названий городов. Затем в цикле **For Each...Next** мы переносим этот список в динамический массив. Далее в цикле **For...Next** мы выводим эти города уже из массива в поля ввода/вывода на HTML-странице. Более подробно объект **Dictionary** рассмотрен в соответствующем разделе.

```
<html>
<head><title>Формы и элементы</title></head>
<SCRIPT LANGUAGE="VBScript">
<!--
sub cmdChange_OnClick
  dim M()                  'Массив для строк
  dim d                    'Список городов
  dim n                    'Количество городов
  set d = CreateObject("Scripting.Dictionary")
  d.Add "0", "Афины"       'Добавить несколько ключей и строк
  d.Add "1", "Белград"
  d.Add "2", "Каир"

  n = 0
  for each i in d          'Заполнить массив элементами списка
    ReDim Preserve M(n+1)
    M(n) = d.Item(i)
    n = n + 1
  next
  for i = 0 To (n-1)       'Вывести строки в поля ввода/вывода
    Document.frmForm.Elements(i).Value = M(i)
  Next
end sub
-->
</SCRIPT>

<body>
<center>
<form NAME="frmForm">
<input Type = "Text"><p>
<input Type = "Text"><p>
<input Type = "Text"><p>
<input Type = "Button" NAME="cmdChange" VALUE="Щёлкните здесь"><p>
</form>
</center>
</body>
</html>
```

2.1.7. Подпрограммы VBScript

Виды подпрограмм

В VBScript есть два вида подпрограмм: процедуры (Sub) и функции (Function).

Процедуры

Процедура в VBScript — это последовательность операторов, заключённых между ключевыми словами **Sub** и **End Sub**. Процедура выполняет некоторые действия, но не возвращает значение (результат выполнения этих действий). Процедура может принимать аргументы (параметры), в качестве которых могут быть константы, переменные или выражения. Если процедура не имеет входных параметров, то после имени процедуры должны стоять пустые круглые скобки.

Процедура, приведённая в следующем примере, использует две распространённых стандартных функции VBScript: MsgBox и InputBox, которые используются для ввода пользователем некоторой информации. Затем отображается результат вычислений, выполненных на основе этой информации. Вычисления выполняются с помощью функции Celsius, созданной в VBScript и рассмотренной далее.

```
Sub ConvertTemp()
  temp = InputBox("Введите температуру по Фаренгейту, F", 1)
  MsgBox "Температура по Цельсию, C =  " & Celsius(temp)
End Sub
```

Функции

Функция в VBScript — это последовательность операторов, заключённых между ключевыми словами **Function** и **End Function**. Функция подобна процедуре, но в отличие от процедуры она может возвращать значение (результат выполнения функции). Функция может принимать аргументы (параметры), в качестве которых могут быть константы, переменные или выражения. Если функция не имеет входных параметров, то после имени функции должны стоять пустые круглые скобки. Если требуется, чтобы функция возвращала значение, то это значение должно присваиваться имени функции в одном или нескольких операторах в теле функции. Функция всегда возвращает данные типа Variant.

Функция, приведённая в следующем примере, используется в примере применения процедур (см. выше). Эта функция преобразует температуру по Фаренгейту в температуру по Цельсию. Функция вызывается в теле процедуры ConvertTemp. Результат вычислений возвращается функцией в вызывающую процедуру.

```
Function Celsius(fDegrees)
  Celsius = (fDegrees - 32) * 5 / 9
End Function
```

Получение данных в процедуру и из процедуры

Каждый элемент данных помещается в вашу процедуру путём передачи параметров (аргументов). Параметры служат как контейнер для данных, которые вы хотите передать в процедуру. Вы можете присваивать параметрам любые имена, отвечающие правилам именования переменных. Когда вы создаёте процедуру или функцию, то после имени подпрограммы должны быть круглые скобки. Все параметры помещаются в круглых скобках после имени подпрограммы. Если параметров несколько, то они разделяются запятой.

Если подпрограмма должна возвращать значение (то есть если вы хотите получить данные из подпрограммы), то вы должны использовать функцию. Помните, что функция может возвращать значение, а процедура — не может. Впрочем, если вы хотите получить какие-либо данные из процедуры, то вы можете использовать для этих целей глобальные переменные. Хотя такой подход не рекомендуется, так как он делает код плохо читаемым и увеличивает вероятность ошибок.

Использование подпрограмм в коде

Функция в вашем коде должна всегда находиться в правой части выражения, то есть результат функции всегда присваивается какой-либо переменной. Например:

```
temp = Celsius(100)
```

или

```
MsgBox "Температура по Цельсию, C =  " & Celsius(temp)
```

Для вызова процедуры из другой подпрограммы вам нужно только напечатать имя процедуры в одной строке с параметрами, передаваемыми в процедуру. Параметры должны разделяться запятой. При вызове процедуры оператор Call использовать не обязательно, но если вы его применяете, то вы должны помещать параметры в круглые скобки.

В следующем примере показан вызов процедуры MyProc двумя способами: с использованием и без использования оператора Call. Оба варианта по сути одинаковы.

```
Call MyProc(firstarg, secondarg)
MyProc firstarg, secondarg
```

Учтите, что при вызове процедуры круглые скобки не применяются, если не используется оператор Call.

Резюме

Функция — это процедура, которая возвращает значение при завершении. Процедура не возвращает значений. В VBScript есть отличия между вызовом процедур и функций.

Если вызывается процедура с параметрами, то параметры не должны заключаться в скобки.

Когда вызывается функция с параметрами, то параметры не должны заключаться в скобки, если возвращаемое значение не используется. Иначе параметры должны заключаться в скобки.

Пример

```
'Это процедура, скобки не используются
Erase ArrayOfNumbers

'Возвращаемое значение используется,
'поэтому параметры заключены в скобки.
Response = MsgBox("Continue?", vbYesNo, "Confirm")

'Возвращаемое значение не используется,
'поэтому параметры не заключены в скобки.
MsgBox "Finished.", vbOKOnly, "Information"
```

2.1.8. Соглашения VBScript

Что такое соглашения?

Соглашения — это советы, которые помогут вам писать сценарии, используя Microsoft Visual Basic Scripting Edition. Соглашения могут включать следующее:

- Соглашения по именованию объектов, переменных и подпрограмм
- Соглашения по созданию комментариев
- Форматирование текста и документирование

Основной смысл использования соглашений — это стандартизация структуры и стиля сценария или набора сценариев таким образом, чтобы вы и другие программисты могли легко читать и понимать ваш код. Использование хорошего стиля программирования позволяет создавать правильный, читабельный и недвусмысленный исходный код, который совместим с соглашениями других языков и является интуитивно понятным.

Соглашения по именованию констант

Ранние версии VBScript не имели механизмов для создания пользовательских констант. Если вы используете константы, то для того, чтобы отличать их от переменных, рекомендуется при именовании констант использовать верхний регистр. Если имя константы состоит из нескольких слов, то слова разделяются символом подчёркивания (_). Пример:

```
USER_LIST_MAX
NEW_LINE
```

Это рекомендуемый способ именования констант. Однако вы можете использовать альтернативную схему именования констант. Поскольку объявление констант выполняется с помощью оператора **Const**, то вы можете именовать константы символами в смешанном регистре с префиксом **con**. Например:

```
conYourOwnConstant
```

Соглашения по именованию переменных

С целью повышения читабельности и совместимости кода, используйте для именования переменных префиксы, приведённые ниже в таблице.

Подтип	Префикс	Пример
Boolean	bln	blnFound
Byte	byt	bytRasterData
Date (Time)	dtm	dtmStart
Double	dbl	dblTolerance
Error	err	errOrderNum
Integer	int	intQuantity
Long	lng	lngDistance
Object	obj	objCurrent
Single	sng	sngAverage
String	str	strFirstName

Область видимости

По возможности переменные должны объявляться как локальные, то есть область видимости переменной должна быть как можно меньше. Переменные в VBScrip имею следующие области видимости:

Область видимости	Где объявлена переменная	Видимость
Уровень процедуры	Событие, функция или процедура	Видима в процедуре, в которой объявлена
Уровень сценария	Раздел HEAD в HTML-странице, вне процедуры	Видима как в процедурах, так и в сценарии

Префиксы переменных

Если сценарий очень большой, то для того, чтобы отличать глобальные переменные от локальных, рекомендуется использовать префиксы. Префикс из одной буквы применяется для глобальных переменных. Локальные переменные именуются без префикса.

Область видимости	Префикс	Пример
Уровень процедуры	Нет	dblVelocity
Уровень сценария	s	sblnCalcInProgress

Описание имён переменных и процедур

Имя переменной или процедуры должно быть содержательным, то есть соответствовать назначению этой переменной или процедуры. Имя может состоять из префикса и символов смешанного регистра. Кроме того имя процедуры должно начинаться с глагола, например InitNameArray или CloseDialog.

Для часто используемых или длинных терминов рекомендуется применять стандартные аббревиатуры. В общем случае имена переменных размером более 32 символов слишком трудны для восприятия. Если используете аббревиатуры, убедитесь, что они содержатся во всём коде сценария. Например, случайный переход от **Cnt** к **Count** может привести к неприятным неожиданностям (это уже рассматривалось в разделе «2.3.2. Переменные VBScript»).

Соглашения по именованию объектов

В приведённой ниже таблице приведён список рекомендуемых соглашений для объектов, с которыми вы можете встретиться при программировании в VBScript.

Тип объекта	Префикс	Пример
3D Panel	pnl	pnlGroup
Animated button	ani	aniMailBox
Check box	chk	chkReadOnly
Combo box, drop-down list box	cbo	cboEnglish
Command button	cmd	cmdExit
Common dialog	dlg	dlgFileOpen
Frame	fra	fraLanguage
Horizontal scroll bar	hsb	hsbVolume
Image	img	imgIcon
Label	lbl	lblHelpMessage
Line	lin	linVertical
List Box	lst	lstPolicyCodes
Spin	spn	spnPages
Text box	txt	txtLastName
Vertical scroll bar	bar	vsbRate
Slider	sld	sldScale

Соглашения о комментариях

Все процедуры должны начинаться с короткого комментария, описывающего назначение процедуры. Этот комментарий не должен описывать алгоритм выполнения процедуры (то есть как процедура работает), потому что процедура может быть со временем изменена и такой подробный комментарий может ввести в заблуждение или, в худшем случае, содержать неверную информацию. Код внутри процедуры снабжается комментариями каждой строки, описывающий выполнение процедуры.

Параметры процедуры должны быть описаны, если их назначение не очевидно. Возвращаемые значения для функции и переменные, значения которых изменяется в процедуре через ссылки на параметры, должны также быть описаны в начале процедуры.

Комментарии, используемые в качестве заголовка процедуры, должны использовать следующие разделы (для примера см. следующий раздел «Форматирование кода»):

Раздел заголовка	Содержание комментария
Назначение	Что делает процедура (но НЕ как делает).
Внешние элементы	Список внешних переменных, элементов управления или других элементов, состояние которых изменяется в процедуре.
Действия	Список действий процедуры над каждой внешней переменной, элементом управления или другим элементом.
Входные параметры	Пояснения для каждого аргумента, назначение которого не является очевидным. Каждый аргумент должен быть описан в отдельной строке.
Возвращаемые значения	Пояснения для возвращаемого значения.

Помните следующие моменты:

- Каждая важная объявленная переменная должна быть описана в отдельной строке комментария.
- Переменные, элементы управления и процедуры должны иметь осмысленные имена, чтобы в комментариях приводились только описания для сложных случаев и в тех ситуациях, когда требуются подробности.
- В начале вашего сценария вы должны привести общее описание сценария, количество объектов, процедур, алгоритмов, диалоговых окон и других элементов. Иногда полезно описание алгоритмов с помощью псевдокода.

Форматирование кода

Экранное пространство должно использоваться экономно насколько это возможно. В то же время при написании исходного текста должна посматриваться логическая структура и вложенность операторов. При этом следует учитывать следующие моменты:

- Стандартные вложенные блоки должны иметь отступы в размере 4 пробелов.
- Общие комментарии процедуры должны быть отделены от кода процедуры одной пустой строкой.
- Высший уровень операторов в теле процедуры должен иметь отступ в 4 пробела, каждый последующий вложенный блок операторов отделяется прибавлением ещё 4 пробелов.

Ниже приведён пример исходного кода, где выполнены указанные здесь рекомендации. В заключение следует отметить, что это лишь РЕКОМЕНДАЦИИ, а не обязательные условия. Вы можете разработать свои принципы форматирования кода. Однако, если вы мечтаете работать в Microsoft (чем чёрт не шутит))), то вам лучше сразу привыкать писать программы по правилам, принятым в Microsoft.

```
'************************************************************
' Назначение: Поиск первого вхождения указанного пользователя
'             в массиве strUserList.
' Входные
' параметры:  strUserList():   список пользователей для поиска.
'             strTargetUser:   имя искомого пользователя.
' Возврат:    Индекс первого найденного элемента массива strUserList,
'             в котором выполнялся поиск строки strTargetUser.
'             Если указанный пользователь не найден,
'             то возвращает -1.
'************************************************************

Function intFindUser (strUserList(), strTargetUser)
    Dim i                       ' Счётчик
    Dim blnFound               ' Флаг завершения поиска
    intFindUser = -1
    i = 0                       ' Инициализация счётчика
    Do While i <= Ubound(strUserList) and Not blnFound
        If strUserList(i) = strTargetUser Then
            blnFound = True     ' Установить флаг в True
            intFindUser = i     ' Вернуть количество циклов
        End If
        i = i + 1               ' Увеличить счётчик на 1
    Loop
End Function
```

2.2. Использование VBScript в Internet Explorer

2.2.1. Простая страница с VBScript

Простая веб-страница

Вы можете создать простую веб-страницу, код которой показан ниже, и проверить её работу в Internet Explorer (другие браузеры не поддерживают VBScript). Если вы щёлкните по кнопке на странице, вы увидите VBScript в действии.

```
<html>
<head><title>Простая веб-страница</title>
<SCRIPT LANGUAGE="VBScript">
<!--
sub Button1_OnClick
  MsgBox "Это достойно того, чтобы увидеть своими глазами"
end sub
-->
</SCRIPT>
</head>
<body>
<h3>Это простая веб-страница</h3><hr>
<form><input NAME="Button1" TYPE="BUTTON" VALUE="Щёлкните здесь"></form>
</body>
</html>
```

Результат выполнения этого сценария не приводит в восторг. Здесь просто выводится на экран диалоговое окно с текстом. В оригинальной документации в этом примере выводится фраза на латинском языке «Mirabile visu», которую я перевёл как смог (возможно, не совсем правильно))). Однако это даёт представление о том, как встраивать сценарии в веб-страницу.

Когда Internet Explorer читает страницу, он находит тег <SCRIPT>, понимает, что здесь находится блок кода VBScript и запоминает этот код. Затем, когда вы щёлкаете по кнопке, Internet Explorer устанавливает связь между кнопкой и кодом и выполняет процедуру.

Процедура, находящаяся в теге <SCRIPT> реагирует на событие. Имя процедуры состоит из двух частей: из имени кнопки **Button_1** и имени события **OnClick** (щелчок по кнопке). Имя кнопки указывается при создании кнопки в параметре NAME тега <INPUT>. Эти два имени объединяются с помощью символа подчёркивания (_). Каждый раз, когда пользователь щёлкает по кнопке, Internet Explorer находит процедуру, связанную с этим событием по имени процедуры (в нашем случае это **Button1_OnClick**) и выполняет сценарий.

Internet Explorer может определить события, связанные с элементами управления формы, которые описаны в документации Internet Explorer Scripting Object Model. Эту документацию можно найти на сайте Microsoft (`http://www.microsoft.com`).

Страницы могут использовать комбинации элементов управления и процедур. В разделе «2.2.2. VBScript и формы» показаны простые случаи взаимодействия между элементами управления.

Другие пути связывания кода с событиями

Несмотря на то, что описанный выше способ является наиболее простым и используется в большинстве случаев, вы можете связать код VBScript с событием другими методами. Internet Explorer позволяет вам добавлять короткий код в строку тега, который создаёт элемент управления. Например, показанный ниже код выполняет те же действия по щелчку на кнопке, что и рассмотренный ранее пример. Этот код встроен в тег <INPUT>.

```
<input NAME="Button2" TYPE="BUTTON" VALUE="И здесь тоже" OnClick='MsgBox
"Выполнен сценарий, код которого находится в теге INPUT"'>
```

Учтите, что вызываемая функция заключается в одинарные кавычки, а строка для функции **MsgBox** заключается в двойные кавычки. Вы можете использовать несколько операторов, разделяя их с помощью двоеточия (:).

Вы можете написать тег <SCRIPT> таким образом, чтобы он применялся только для определённого события и для указанного элемента управления:

```
<SCRIPT LANGUAGE="VBScript" EVENT="OnClick" FOR="Button3">
<!--
    MsgBox "Третья кнопка"
-->
</SCRIPT>
```

Поскольку в теге <SCRIPT> уже определены событие и элемент управления, то операторы **Sub** и **End Sub** использовать не нужно.

2.2.2. VBScript и формы

Простая проверка

Вы можете использовать сценарии VBScript для отправки данных пользователя на сервер. Вы также можете выполнять какие-либо вычисления без отправки данных на сервер.

Ниже приведён простой пример проверки на стороне клиента. На веб-странице расположены поле для ввода текста и кнопка.

В отличие от примера, приведённого в разделе «2.4.1. Простая страница с VBScript», здесь свойство **Value** текстового поля используется для проверки введённого значения. Чтобы получить значение текстового поля, в коде используется ссылка на имя этого поля. То есть в сценарии к этому полю мы обращаемся по полному имени, которое состоит из имени формы и имени элемента управления. Кроме этого в имени ещё используется объект **Document**, который связан с текущей веб-страницей. В принципе, слово **Document** в имени можно опустить. Итак, полное имя текстового поля, которое можно использовать для получения содержимого этого элемента, выглядит так:

```
Document.ValidForm.Text1
```

Но чтобы немного сократить это имя, мы сначала получаем ссылку на форму в переменную TheForm и затем уже используем эту переменную. Для того, чтобы получить ссылку на форму, мы используем оператор **Set**, который связывает форму с переменной TheForm:

```
set TheForm = Document.ValidForm
```

Пример веб-страницы:

```
<html>
<head><title>Простая проверка</title>
<SCRIPT LANGUAGE="VBScript">
<!--
sub Button1_OnClick
  dim TheForm
  set TheForm = Document.ValidForm
  if IsNumeric(TheForm.Text1.Value) then
    if TheForm.Text1.Value < 1 or TheForm.Text1.Value > 10 then
      MsgBox "Введите число от 1 до 10"
    else
      MsgBox "Спасибо"
    end if
  else
    MsgBox "Пожалуйста введите число"
  end if
end sub
-->
</SCRIPT>
</head>
<body>
<h3>Простая проверка</h3><hr>
<form NAME="ValidForm">
Введите число от 1 до 10:
<input NAME="Text1" TYPE="TEXT" SIZE="2">
<input NAME="Button1" TYPE="BUTTON" VALUE="OK">
</form>
</body>
</html>
```

Использование числовых значений

В приведённом выше примере значение проверяется как число. Для этого используется функция **IsNumeric**, которая возвращает TRUE, если переданный в неё параметр является числом. Если это текст, то возвращается значение FALSE. Таким образом мы выполняем проверку введённых пользователем данных. Если пользователь ошибся и ввёл текст вместо числа, то ему выдаётся сообщение «Пожалуйста введите число».

Несмотря на то, что VBScript автоматически преобразует строки и числа, всегда рекомендуется проверять введённые пользователем данные на предмет принадлежности к тому или иному подтипу и при необходимости использовать функции преобразования.

При выполнении математических операций с введёнными в текстовые поля значениями всегда преобразовывайте эти значения в числовой подтип, потому что оператор + (плюс) может использоваться как для сложения чисел, так и для соединения строк. Например, если в поле Text1 введено 1, а в поле Text2 введено 2, то вы получите следующий результат:

```
A = Text1.Value + Text2.Value         ' Здесь A будет равно строке "12"
A = CDbl(Text1.Value) + Text2.Value   ' Здесь A будет равно числу 3
```

Проверка и отправка данных на сервер

В приведённом выше примере используется обычная кнопка. Если используется кнопка типа Submit, то в этом примере мы бы не увидели данных для проверки, данные ушли бы непосредственно на сервер. Отказ от использования Submit позволит вам проверять данные, но не позволит отправлять данные на сервер. Чтобы решить эту проблему (то есть иметь возможность отправки данных на сервер при использовании обычной кнопки), вам нужно добавить одну строку в код сценария:

```
<SCRIPT LANGUAGE="VBScript">
<!--
Sub Button2_OnClick
  Dim TheForm
  Set TheForm = Document.ValidForm
  If IsNumeric(TheForm.Text1.Value) Then
    If TheForm.Text1.Value < 1 Or TheForm.Text1.Value > 10 Then
      MsgBox "Введите число от 1 до 10"
    Else
      MsgBox "Спасибо"
      TheForm.Submit  ' Данные верны, отправить на сервер.
    End If
  Else
    MsgBox "Пожалуйста введите число"
  End If
End Sub
-->
</SCRIPT>
```

Отправку данных на сервер выполняет метод **Submit** объекта **Form**. После этого сервер обрабатывает данные. Более подробную информацию о методе **Submit** и других методах вы можете найти в документации Internet Explorer Scripting Object Model. Эту документацию можно найти на сайте Microsoft (`http://www.microsoft.com`).

Здесь вы увидели только стандартные объекты HTML <FORM>. Кроме этого Internet Explorer позволяет вам разрабатывать полноценные мощные элементы управления ActiveX (ранее они назывались OLE) и Java объекты.

2.2.3. Использование VBScript с объектами

Использование объектов

Независимо от того, используете вы элементы ActiveX (ранее они назывались OLE) и Java объекты, Microsoft Visual Basic Scripting Edition и Internet Explorer обрабатывают их одинаково. Если вы используете Internet Explorer с установленным элементом управления **Label**, вы можете проверить работу следующего кода.

Вы включаете объект, используя тег <OBJECT> и устанавливаете его начальные свойства в теге <PARAMS>. Если вы являетесь программистом Visual Basic, то вы должны знать, то тег <PARAMS> используется только для установки начальных свойств элемента управления на форме.

Например, следующий фрагмент кода добавляет на страницу элемент управления ActiveX Label:

```
<object
  classid="clsid:99B42120-6EC7-11CF-A6C7-00AA00A47DD2"
  id=lblActiveLbl
  width=250
  height=250
  align=left
  hspace=20
  vspace=0
>
<param NAME="Angle" VALUE="90">
<param NAME="Alignment" VALUE="4">
<param NAME="BackStyle" VALUE="0">
<param NAME="Caption" VALUE="A Simple Desultory Label">
<param NAME="FontName" VALUE="Verdana, Arial, Helvetica">
<param NAME="FontSize" VALUE="20">
<param NAME="FontBold" VALUE="1">
<param NAME="FrColor" VALUE="0">
</object>
```

Вы можете получить свойства, установить свойства и выполнить методы объекта также как и для любого другого элемента управления формы. В следующем примере, включив в код тег <FORM>, вы можете манипулировать двумя свойствами элемента управления **Label**.

```
<form NAME="LabelControls">
<input TYPE="TEXT" NAME="txtNewText" SIZE=25>
<input TYPE="BUTTON" NAME="cmdChangeIt" VALUE="Измениь текст">
<input TYPE="BUTTON" NAME="cmdRotate" VALUE="Повернуть">
</form>
```

Событие, связанное с кнопкой **cmdChangeIt**, изменяет текст элемента **Label**:

```
<SCRIPT LANGUAGE="VBScript">
<!--
sub cmdChangeIt_onClick
  dim TheForm
  set TheForm = Document.LabelControls
  lblActiveLbl.Caption = TheForm.txtNewText.Value
end sub
-->
</SCRIPT>
```

Связь элементов управления, находящихся внутри формы и значений полей ввода/вывода выполняется также, как это описано в разделе «2.4.2. VBScript и формы».

Несколько элементов ActiveX доступны для использования в Internet Explorer. Вы можете найти полную информацию о свойствах, методах и событиях, а также об идентификаторах классов (CLSID) для элементов управления на сайте Microsoft (`http://www.microsoft.com`). Вы можете найти больше информации о теге <OBJECT> на странице Internet Explorer 4.0 Author's Guide and HTML Reference.

ПРИМЕЧАНИЕ
Простая реализация Internet Explorer требует наличия фигурных скобок ({}) вокруг атрибутов класса, что не соответствует спецификации W3C. Использование фигурных скобок в текущей реализации генерирует сообщение «This page uses an outdated version of the <OBJECT> tag»

Боюсь, что всё вышесказанное, для многих из вас не очень понятно, особенно если вы раньше никогда не использовали тег <OBJECT>. В таком случае лучше всё проверить на практике. Однако. Скорее всего у вас (как и у меня), элемент управления **Label** не установлен, поэтому приведённый здесь пример работать не будет. Чтобы исправить это недоразумение, приведу постой пример использования описанного в этом разделе подхода. Отличие будет заключаться в том, что в качестве объекта в моем примере будет использоваться тег <IMG>.

```
<html>
<head><title>Простая проверка</title>
<SCRIPT LANGUAGE="VBScript">
<!--
sub cmdPlusSize_onClick
     imgTest.Width = imgTest.Width * 1.1
     imgTest.height = imgTest.height * 1.1
end sub
sub cmdMinusSize_onClick
     imgTest.Width = imgTest.Width / 1.1
     imgTest.height = imgTest.height / 1.1
end sub
-->
</SCRIPT>
</head>
<body>
<h3>Работа с объектами</h3><hr>

<img
  border="0"
  src="vbs-error.jpg"
  width="542"
  height="204"
  id=imgTest
>

<form NAME="LabelControls">
<input TYPE="BUTTON" NAME="cmdPlusSize" VALUE="Увеличить на 10%">
<input TYPE="BUTTON" NAME="cmdMinusSize" VALUE="Уменьшить на 10%">
</form>
</body>
</html>
```

Здесь одна кнопка увеличивает размер картинки на 10% при каждом щелчке, а другая кнопка — уменьшает. Ну а всё остальное вам должно быть понятно, если вы читали предыдущие разделы.

3. ИСПОЛЬЗОВАНИЕ ОБЪЕКТА FILESYSTEMOBJECT

3.1. Модель объекта FileSystemObject

При разработке сценариев для ASP, Windows Scripting Host или других приложений, где могут быть использованы сценарии, часто важно иметь возможность создавать, удалять, изменять папки (директории) и файлы на веб-сервере. Также может возникнуть необходимость в получении информации о дисках, установленных на сервере.

Сценарии позволяют вам управлять дисками, папками и файлами, используя объектную модель FileSystemObject (FSO), которая описана в следующих разделах.

3.2. Введение в FileSystemObject и Scripting Run-Time Library Reference

Объектная модель FileSystemObject (FSO) позволяет вам привычный синтаксис **Объект.Метод** с богатым набором свойств, методов и событий для работы с папками и файлами.

Эти основанные на объектах инструменты используются с:

- **HTML** для создания веб-страниц
- **Windows Scripting Host** для создания пакетных файлов для Windows
- **Script Control** для предоставления совместимости сценариев в приложениях, разработанных на других языках

Поскольку использование FSO на стороне клиента влечёт за собой серьёзные угрозы безопасности в виде потенциальной возможности получить несанкционированный доступ к локальной файловой системе клиента, в этой документации рассматривается использование объектной модели FSO для создания сценариев, выполняемых в веб-страницах на стороне сервера. Так как по умолчанию используется сторона сервера, Internet Explorer сохраняет настройки безопасности таким образом, чтобы не позволить использование объекта FileSystemObject на стороне клиента. Изменение этих настроек на локальном компьютере разрешает несанкционированный доступ к файловой системе, что потенциально может повлечь за собой разрушение файловой системы и потерю данных.

Объектная модель FSO даёт возможность вашему приложению на стороне сервера создавать, переименовывать, перемещать и удалять папки, а также определять их местоположение и проверять, существует ли папка. Вы можете также получить информацию о папке, такую как имя, дату создания и последнего изменения и т.п.

Объектная модель FSO делает такой же лёгкой работу с файлами. При обработке файлов наиболее часто возникают задачи записи и чтения данных. Вы можете создавать файлы, вставлять, читать и изменять данные. Обычно данные записываются в базу данных, такую как Access или SQL Server, добавляются в заголовок вашего приложения, записываются в двоичные или текстовые файлы. Может случиться так, что вы не будете знать форматов некоторых файлов или доступ к данным потребует полного набора функций, связанного с полнофункциональной базой данных.

Объектная модель FSO содержится в Библиотеке типов — Scripting type library (Scrrun.dll), которая поддерживает создание текстовых файлов и управление ими с помощью объекта **TextStream**. Работа с бинарными файлами пока не поддерживается, но это планируется осуществить в будущих версиях.

3.3. Объекты FileSystemObject

Объектная модель FileSystemObject (FSO) содержит следующие объекты и коллекции:

Объект/Коллекция	Описание
FileSystemObject	Основной объект. Содержит методы и свойства, которые позволяют вам создавать, удалять, получать информацию и управлять дисками, папками и файлами. Многие методы, связанные с этим объектом, дублируют его в других объектах FSO (это сделано для совместимости).
Drive	Объект. Содержит методы и свойства, позволяющие вам получать информацию о дисках, имеющихся в системе. Учтите, что «Drive» - это не обязательно жёсткий диск. Это может быть CD-ROM, флеш-диск и т.п. Диск не обязательно должен быть физически подключен к системе. Это может быть логически подключенный сетевой диск.
Drives	Коллекция. Предоставляет список имеющихся в системе дисков, как физических, так и логических. Эта коллекция включает в себя все диски, независимо от типа. Съёмные медиа-диски не обязательно должны быть вставлены, чтобы попасть в эту коллекцию (то есть вам не обязательно вставлять компакт-диск в CD-ROM, чтобы диск, связанный с этим устройством появился в этой коллекции).
File	Объект. Содержит методы и свойства, которые позволяют создавать, удалять и перемещать файлы. Также возможно получить имя файла, путь к файлу и другие свойства.
Files	Коллекция. Предоставляет список всех файлов, содержащихся в папке.
Folder	Объект. Содержит методы и свойства, которые позволяют создавать, удалять и перемещать папки. Также возможно получить имя папки, путь к папке и другие свойства.
Folders	Коллекция. Предоставляет список всех папок, имеющихся в объекте **Folder**.
TextStream	Объект. Позволяет читать и записывать текстовые файлы.

3.4. Программирование FileSystemObject

Для программирования с использованием объектной модели FileSystemObject (FSO):

- Используйте метод CreateObject для создания объекта FileSystemObject
- Используйте нужный метод вновь созданного объекта
- Используйте свойства объекта

Объектная модель FSO содержится в Scripting type library, которая находится в файле Scrrun.dll. Поэтому вы должны иметь этот файл в соответствующей системной директории на вашем веб-сервере, чтобы использовать FileSystemObject.

Создание объекта FileSystemObject

Сначала нужно создать объект FileSystemObject, используя метод CreateObject. В VBScript используйте для примера следующий код:

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

Scripting — это имя библиотеки типов, а **FileSystemObject** — это имя объекта, который нужно создать. Вы можете создать только один экземпляр объекта FileSystemObject, независимо от того, как много раз вы будете пытаться создать другие объекты.

Использование подходящих методов

Далее вы можете использовать нужные методы объекта FileSystemObject. Например, для создания нового объекта, используйте метод **CreateTextFile** или **CreateFolder** (в объектной модели FSO не поддерживается создание или удаление дисков).

Для удаления объекта используйте методы **DeleteFile** и **DeleteFolder** объекта FileSystemObject. Вы можете также копировать и перемещать файлы и папки, используя соответствующие методы.

ПРИМЕЧАНИЕ
Некоторые функции в объекте FileSystemObject являются излишними. Например, вы можете использовать для копирования файла как метод **CopyFile** объекта FileSystemObject, так и метод **Copy** объекта File. Методы работают одинаково, но предоставлены для обеспечения гибкости программирования.

Доступ к существующим дискам, файлам и папкам

Чтобы получить доступ к дискам, папкам и файлам, используйте соответствующий **get** метод объекта FileSystemObject:

- **GetDrive**
- **GetFolder**
- **GetFile**

Пример организация доступа к существующему файлу в VBScript:

```
Dim fso, f1
Set fso = CreateObject("Scripting.FileSystemObject")
Set f1 = fso.GetFile("c:\test.txt")
```

Не используйте **get** метод для вновь создаваемых объектов, так как функции **create** уже возвращают дескриптор объекта. Например, если вы создаёте новую папку, используя метод **CreateFolder**, не используйте метод **GetFolder** для доступа к её свойствам, таким как имя, путь и размер. Просто присвойте результат выполнения метода **CreateFolder** переменной, чтобы получить дескриптор вновь созданной папки, а затем получите её свойства, методы и события. Пример приведён ниже:

```
Sub CreateFolder
  Dim fso, fldr
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set fldr = fso.CreateFolder("C:\MyTest")
End Sub
```

Доступ к свойствам объекта

Однажды получив дескриптор объекта, вы можете получить его свойства. Например, чтобы получить имя конкретной папки, сначала создайте экземпляр объекта, затем получите дескриптор на объект с помощью соответствующего метода (в нашем случае это метод **GetFolder**, так как папка уже существует).

Используйте этот код для получения дескриптора с помощью метода **GetFolder**:

```
Set fldr = fso.GetFolder("c:\Windows")
```

Теперь, когда вы имеете дескриптор объекта **Folder**, вы можете проверить свойство **Name** (Имя):

```
MsgBox "Имя папки: " & fldr.Name
```

Чтобы получить время последнего изменения файла, используйте следующий синтаксис:

```
Dim fso, f1
' Получить объект File.
Set f1 = fso.GetFile("c:\Windows\desktop.ini")
' Вывести информацию.
MsgBox "Время последнего изменения файла: " & f1.DateLastModified
```

3.5. Работа с дисками и папками

Объектная модель FileSystemObject (FSO) позволяет вам работать программно с дисками и файлами так же, как вы это делаете в Проводнике Windows интерактивно. Вы можете копировать и перемещать папки, получать информацию о дисках и папках и т.п.

Получение информации о дисках

Объект **Drive** позволяет вам получать информацию о дисках, имеющихся в системе. Причём это могут быть как физические диски, так и сетевые. Их свойства позволяют получить следующую информацию:

- Общий объём диска в байтах (свойство **TotalSize**)
- Объём свободного места на диске в байтах (свойства **AvailableSpace** или **FreeSpace**)
- Какая буква назначена диску (свойство **DriveLetter**)
- Тип диска: съёмный, не съёмный, сетевой, CD-ROM или RAM-диск (свойство **DriveType**)
- Серийный номер диска (свойство **SerialNumber**)
- Тип файловой системы: FAT, FAT32, NTFS или др. (свойство **FileSystem**)
- Доступность диска в текущий момент (свойство **IsReady**)
- Сетевое имя или метка тома (свойства **ShareName** и **VolumeName**)
- Путь к корневой директории диска (свойства **Path** и **RootFolder**)

Пример использования данного объекта см. в разделе «3.7. Пример работы с объектом FileSystemObject».

Пример использования объекта Drive

Используйте объект **Drive** для получения информации о диске. Вы не увидите ссылку на актуальный объект **Drive** в следующем коде. Вместо этого используется метод **GetDrive** для получения ссылки на существующий объект **Drive** (в нашем примере это переменная **drv**).

Следующий пример демонстрирует использование объекта **Drive** в VBScript:

```
Sub ShowDriveInfo(drvPath)
  Dim fso, drv, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set drv = fso.GetDrive(fso.GetDriveName(drvPath))
  s = "Диск " & UCase(drvPath) & " - "
  s = s & drv.VolumeName
  s = s & ", Общий объём: " & FormatNumber(drv.TotalSize / 1024, 0)
  s = s & " Kb"
  s = s & ", Свободное место: " & FormatNumber(drv.FreeSpace / 1024, 0)
  s = s & " Kb"
  MsgBox s
End Sub

ShowDriveInfo("C:")
```

Работа с папками

Общие задачи по работе с папками и методы для их выполнения описаны ниже в таблице:

Задача	**Метод**
Создание папки	FileSystemObject.CreateFolder
Удаление папки	Folder.Delete или FileSystemObject.DeleteFolder
Перемещение папки	Folder.Move или FileSystemObject.MoveFolder
Копирование папки	Folder.Copy или FileSystemObject.CopyFolder
Получение имени папки	Folder.Name
Узнать, существует ли папка на диске	FileSystemObject.FolderExists
Получить экземпляр существующего объекта **Folder**	FileSystemObject.GetFolder
Получить имя родительской папки	FileSystemObject.GetParentFolderName
Получить путь системных папок	FileSystemObject.GetSpecialFolder

Пример использования данного объекта см. в разделе «3.7. Пример работы с объектом FileSystemObject».

В следующем примере показано использование объектов **Folder** и **FileSystemObject**, где выполняется управление папками и получение информации о них:

```
Sub ShowFolderInfo()
  Dim fso, fldr, s
  ' Получить экземпляр FileSystemObject.
  Set fso = CreateObject("Scripting.FileSystemObject")
  ' Получить объект Drive.
  Set fldr = fso.GetFolder("c:")
  ' Вывести имя родителя.
  MsgBox "Имя родителя: " & fldr
  ' Вывести имя диска.
  MsgBox "Диск " & fldr.Drive
  ' Вывести имя корневого каталога.
  If fldr.IsRootFolder = True Then
    MsgBox "Это корневой каталог"
  Else
    MsgBox "Это не корневой каталог"
  End If
  ' Создать новую папку
  fso.CreateFolder ("C:\Bogus")
  MsgBox "Создана папка C:\Bogus"
  ' Основное имя папки
  MsgBox "Имя папки = " & fso.GetBaseName("c:\bogus")
  ' Удалить созданную папку
  fso.DeleteFolder ("C:\Bogus")
  MsgBox "Удалена папка C:\Bogus"
End Sub
```

3.6. Работа с файлами

Имеются две основных категории операций с файлами:

- Создание, добавление или удаление данных и чтение файлов
- Перемещение, копирование и удаление файлов

Создание файлов

Есть три способа создания пустого текстового файла (иногда называемого как «text stream»).

Первый способ — это использование метода **CreateTextFile**. В следующем примере показано, как создать текстовый файл, используя этот метод:

```
Dim fso, f1
Set fso = CreateObject("Scripting.FileSystemObject")
Set f1 = fso.CreateTextFile("c:\testfile.txt", True)
```

Пример использования данного метода см. в разделе «3.7. Пример работы с объектом FileSystemObject».

Второй способ — это использование метода **OpenTextFile** объекта **FileSystemObject** с набором флагов **ForWriting**. В следующем примере показано, как создать текстовый файл, используя этот метод:

```
Dim fso, ts
Const ForWriting = 2
Set fso = CreateObject("Scripting. FileSystemObject")
Set ts = fso.OpenTextFile("c:\test.txt", ForWriting, True)
```

Третий способ — это использование метода **OpenAsTextStream** с набором флагов **ForWriting**. В следующем примере показано, как создать текстовый файл, используя этот метод:

```
Dim fso, f1, ts
Const ForWriting = 2
Set fso = CreateObject("Scripting.FileSystemObject")
fso.CreateTextFile ("c:\test1.txt")
Set f1 = fso.GetFile("c:\test1.txt")
Set ts = f1.OpenAsTextStream(ForWriting, True)
```

Добавление данных в файл

Однажды создав текстовый файл, вы можете добавлять в него данные, для чего нужно выполнить три действия:

- Открыть текстовый файл
- Записать данные
- Закрыть файл

Для открытия существующего файла используйте метод **OpenTextFile** объекта **FileSystemObject** или метод **OpenAsTextStream** объекта **File**.

Для записи данных в текстовый файл используйте методы **Write**, **WriteLine** или **WriteBlankLines** объекта **TextStream**, в зависимости от задач, описанных в приведённой ниже таблице:

Задача	Метод
Запись данных в текстовый файл без символа перехода на новую строку в конце	Write
Запись данных в текстовый файл с символом перехода на новую строку в конце	WriteLine
Запись одной или более пустых строк в открытый текстовый файл	WriteBlankLines

Для закрытия текстового файла используйте метод **Close** объекта **TextStream**. Закрыть файл можно также методом **Close** объекта **FileSystemObject**. Пример использования данных методов см. в разделе «3.7. Пример работы с объектом FileSystemObject».

ПРИМЕЧАНИЕ
Символ новой строки содержит символ или символы (зависит от операционной системы) для перевода курсора в начало новой строки (возврат каретки/перевод строки). Следует учитывать, что некоторые строки уже могут иметь в конце эти непечатаемые символы.

В следующем примере показано, как выполняется запись данных в открытый файл всеми тремя методами, затем файл закрывается:

```
Sub CreateFile()
  Dim fso, tf
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set tf = fso.CreateTextFile("c:\testwritefile.txt", True)
  ' Записать строку с переводом на новую строку.
  tf.WriteLine("Тестирование 1, 2, 3.")
  ' Записать три пустых строки в файл.
  tf.WriteBlankLines(3)
  ' Записать строку
  tf.Write ("Это тест")
  tf.Close
End Sub
```

Чтение файлов

Для чтения данных из файла используйте методы **Read, ReadLine** или **ReadAll** объекта **TextStream**. Эти методы и задачи, решаемые с их помощью, описаны в приведённой ниже таблице:

Задача	Метод
Чтение указанного количества символов из файла	Read
Чтение полной строки из файла (до символа конца строки, но не включая его)	ReadLine
Чтение всего содержимого файла	ReadAll

Пример использования данных методов см. в разделе «3.7. Пример работы с объектом FileSystemObject».

Если вы используете методы **Read** или **ReadLine** и хотите пропустить определённую часть данных, то используйте методы **Scip** или **ScipLine**. Полученный в результате работы этих методов текст может быть сохранён в строку, которую можно отобразить на экране в элементе управления, передать в качестве параметра в строковую функцию (например, в **Left**, **Right** или **Mid**), соединить с другой строкой и т.п.

В следующем примере показано, как открыть файл, записать в него данные, а затем прочитать их:

```
Sub ReadFiles
  Dim fso, f1, ts, s
  Const ForReading = 1
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f1 = fso.CreateTextFile("c:\testreadfile.txt", True)
  ' Записать строку
  f1.WriteLine "Hello World"
  f1.WriteBlankLines(1)
  f1.Close
  ' Прочитать содержимое файла
  Set ts = fso.OpenTextFile("c:\testreadfile.txt", ForReading)
  s = ts.ReadLine
  MsgBox "Содержимое файла = '" & s & "'"
  ts.Close
End Sub
```

Перемещение, копирование и удаление файлов

Объектная модель FSO имеет для каждой из операций перемещения, копирования или удаления файлов по два метода. Эти методы и задачи, решаемые с их помощью, описаны в приведённой ниже таблице:

Задача	Метод
Перемещение файла	**File.Move** или **FileSystemObject.MoveFile**
Копирование файла	**File.Copy** или **FileSystemObject.CopyFile**
Удаление файла	**File.Delete** или **FileSystemObject.DeleteFile**

Пример использования данных методов см. в разделе «3.7. Пример работы с объектом FileSystemObject».

В следующем примере создаётся текстовый файл в корневом каталоге диска C, записывается в него некоторая информация. Затем файл перемещается в директорию с именем \tmp, копируется в директорию \temp, а затем удаляются копии из обеих директорий.

Для проверки работы этого примера создайте в корневом каталоге диска C директории **tmp** и **temp**.

```
Sub ManipFiles
  Dim fso, f1, f2, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f1 = fso.CreateTextFile("c:\testfile.txt", True)
  ' Записать строку
  f1.Write ("This is a test.")
  ' Закрыть файл для записи
  f1.Close
  ' Получить дескриптор файла в корневом каталоге C:\.
  Set f2 = fso.GetFile("c:\testfile.txt")
  ' Переместить файл в каталог \tmp
  f2.Move ("c:\tmp\testfile.txt")
  ' Копировать файл в каталог \temp.
  f2.Copy ("c:\temp\testfile.txt")
  ' Получить дескрипторы файлов
  Set f2 = fso.GetFile("c:\tmp\testfile.txt")
  Set f3 = fso.GetFile("c:\temp\testfile.txt")
  ' Удалить файлы
  f2.Delete
  f3.Delete
  MsgBox "Все операции выполнены!"
End Sub
```

3.7. Пример работы с объектом FileSystemObject

Пример кода, описанный в этом разделе предоставляет реально работающий пример, демонстрирующий множество доступных в объекте FileSystemObject функций. В этом коде показано, как можно совместно использовать эти функции и как можно эффективно работать с этими функциями в вашем коде.

Учтите, что код написан правильно, однако некоторые участки кода могут нуждаться в доработке под ваш конкретный компьютер. Эти доработки в основном сводятся к изменениям путей к файлам и различий между Active Server Pages и Windows Scripting Host.

Чтобы выполнить этот код на Active Server Pages, используйте следующие шаги:

1. Создайте стандартную веб-страницу с расширением .asp
2. Скопируйте приведённый здесь код в файл между тегами <BODY>...</BODY>
3. Заключите весь код в теги <%...%>
4. Переместите оператор Option Explicit из текущей позиции кода в самый верх страницы, даже перед тегом <HTML>
5. Поместите оператор Option Explicit в теги <%...%> и убедитесь, что он выполняется на стороне сервера
6. Добавьте следующий код в конец представленного здесь примера:

```
Sub Print(x)
  Response.Write "<PRE><FONT FACE=""Courier New"" SIZE=""1"">"
  Response.Write x
  Response.Write "</FONT></PRE>"
End Sub
Main
```

Представленный выше код добавляет процедуру печати, которая будет выполняться на стороне сервера, но отображать результат на стороне клиента. Для выполнения этого кода на Windows Scripting Host в конец примера нужно добавить вместо указанного выше следующий код:

```
Sub Print(x)
  WScript.Echo x
End Sub
Main
```

А теперь непосредственно пример использования функций объекта FileSystemObject:

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'
' Пример кода FileSystemObject
'
' Copyright 1998 Microsoft Corporation.  All Rights Reserved.
'
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Option Explicit
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'
' Особенности чтения кода:
'
' 1) В этом коде для строковых операций конкатенации всегда используется
'    оператор "&". Так как строковые операции отнимают много ресурсов, этот путь
'    является неэффективным для написания кода. Однако этот способ вполне
'    работоспособен и может использоваться, потому что эта программа использует
'    дисковые операции, а дисковые операции выполняются существенно медленнее,
'    чем операции с памятью, используемые для конкатенации строк. Имейте в виду,
'    что это демонстрационный, а не реальный код.
'
' 2) "Option Explicit" использован, так как доступ к объявленным переменным выполняется
'    несколько быстрее, чем к необъявленным. Это также снижает вероятность ошибок
'    в вашем коде, таких, например, как печать DriveTypeCDORM вместо DriveTypeCDROM.
'
' 3) В этом коде отсутствует обработка ошибок, чтобы сделать код более читабельным.
'    Несмотря на то, что определённые меры предосторожности приняты, убедитесь, что
'    этот код не навредит вашей файловой системе. В реальном коде используйте
'    On Error Resume Next и объект Err для предотвращения возможных ошибок.
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'
' Некоторые доступные глобальные переменные
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim TabStop
Dim NewLine

Const TestDrive = "C"
Const TestFilePath = "C:\Test"

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'
' Константы, требуемые для Drive.DriveType
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Const DriveTypeRemovable = 1
Const DriveTypeFixed = 2
Const DriveTypeNetwork = 3
Const DriveTypeCDROM = 4
Const DriveTypeRAMDisk = 5

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'
' Константы, требуемые для File.Attributes
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Const FileAttrNormal  = 0
Const FileAttrReadOnly = 1
Const FileAttrHidden = 2
Const FileAttrSystem = 4
Const FileAttrVolume = 8
Const FileAttrDirectory = 16
Const FileAttrArchive = 32
Const FileAttrAlias = 64
Const FileAttrCompressed = 128
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'
' Константы для открытия файлов
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Const OpenFileForReading = 1
Const OpenFileForWriting = 2
Const OpenFileForAppending = 8
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'
' ShowDriveType
'
' Назначение:
'
' Генерация строки, описывающей тип диска и получение объекта Drive.
'
' Демонстрируется следующее
'
' - Drive.DriveType
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Function ShowDriveType(Drive)

	Dim S

	Select Case Drive.DriveType
	Case DriveTypeRemovable
		S = "Съёмный"
	Case DriveTypeFixed
		S = "Несъёмный"
	Case DriveTypeNetwork
		S = "Сетевой"
	Case DriveTypeCDROM
		S = "CD-ROM"
	Case DriveTypeRAMDisk
		S = "RAM-диск"
	Case Else
		S = "Неизвестный"
	End Select

	ShowDriveType = S

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'
' ShowFileAttr
'
' Назначение:
'
' Генерация строки, описывающей атрибуты файла или папки.
'
' Демонстрируется следующее
'
' - File.Attributes
' - Folder.Attributes
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Function ShowFileAttr(File) ' File - файл или папка

	Dim S
	Dim Attr

	Attr = File.Attributes

	If Attr = 0 Then
		ShowFileAttr = "Normal"
		Exit Function
	End If

	If Attr And FileAttrDirectory  Then S = S & "Каталог "
	If Attr And FileAttrReadOnly   Then S = S & "Только чтение "
	If Attr And FileAttrHidden     Then S = S & "Скрытый "
	If Attr And FileAttrSystem     Then S = S & "Системный "
	If Attr And FileAttrVolume     Then S = S & "Том "
	If Attr And FileAttrArchive    Then S = S & "Архивный "
	If Attr And FileAttrAlias      Then S = S & "Псевдоним "
	If Attr And FileAttrCompressed Then S = S & "Сжатый "

	ShowFileAttr = S

End Function
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'
' GenerateDriveInformation
'
' Назначение:
'
' Генерация строки, описывающей текущее состояние доступных дисков.
'
' Демонстрируется следующее
'
' - FileSystemObject.Drives
' - Iterating the Drives collection
' - Drives.Count
' - Drive.AvailableSpace
' - Drive.DriveLetter
' - Drive.DriveType
' - Drive.FileSystem
' - Drive.FreeSpace
' - Drive.IsReady
' - Drive.Path
' - Drive.SerialNumber
' - Drive.ShareName
' - Drive.TotalSize
' - Drive.VolumeName
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Function GenerateDriveInformation(FSO)

        Dim Drives
        Dim Drive
        Dim S

        Set Drives = FSO.Drives

        S = "Количество дисков:" & TabStop & Drives.Count & NewLine & NewLine

        ' Создание первой строки отчёта.
        S = S & String(2, TabStop) & "Диск"
        S = S & String(3, TabStop) & "Файл"
        S = S & TabStop & "Всего"
        S = S & TabStop & "Свободно"
        S = S & TabStop & "Доступно"
        S = S & TabStop & "Серийный" & NewLine

        ' Создание второй строки отчёта.
        S = S & "Буква"
        S = S & TabStop & "Путь"
        S = S & TabStop & "Тип"
        S = S & TabStop & "Готовность"
        S = S & TabStop & "Имя"
        S = S & TabStop & "Система"
        S = S & TabStop & "Пространство"
        S = S & TabStop & "Пространство"
        S = S & TabStop & "Пространство"
        S = S & TabStop & "Номер" & NewLine

        ' Разделительная линия.
        S = S & String(105, "-") & NewLine
```

```
	For Each Drive In Drives

		S = S & Drive.DriveLetter
		S = S & TabStop & Drive.Path
		S = S & TabStop & ShowDriveType(Drive)
		S = S & TabStop & Drive.IsReady

		If Drive.IsReady Then
			If DriveTypeNetwork = Drive.DriveType Then
				S = S & TabStop & Drive.ShareName
			Else
				S = S & TabStop & Drive.VolumeName
			End If

			S = S & TabStop & Drive.FileSystem
			S = S & TabStop & Drive.TotalSize
			S = S & TabStop & Drive.FreeSpace
			S = S & TabStop & Drive.AvailableSpace
			S = S & TabStop & Hex(Drive.SerialNumber)

		End If

		S = S & NewLine

	Next

	GenerateDriveInformation = S

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'
' GenerateFileInformation
'
' Назначение:
'
' Генерация строки, описывающей текущее состояние файла.
'
' Демонстрируется следующее
'
' - File.Path
' - File.Name
' - File.Type
' - File.DateCreated
' - File.DateLastAccessed
' - File.DateLastModified
' - File.Size
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Function GenerateFileInformation(File)

	Dim S

	S = NewLine & "Путь:" & TabStop & File.Path
	S = S & NewLine & "Имя:" & TabStop & File.Name
	S = S & NewLine & "Тип:" & TabStop & File.Type
	S = S & NewLine & "Атрибуты:" & TabStop & ShowFileAttr(File)
	S = S & NewLine & "Создан:" & TabStop & File.DateCreated
	S = S & NewLine & "Доступен:" & TabStop & File.DateLastAccessed
	S = S & NewLine & "Изменён:" & TabStop & File.DateLastModified
	S = S & NewLine & "Размер" & TabStop & File.Size & NewLine

	GenerateFileInformation = S

End Function
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'
' GenerateFolderInformation
'
' Назначение:
'
' Генерация строки, описывающей текущее состояние папки.
'
' Демонстрируется следующее
'
' - Folder.Path
' - Folder.Name
' - Folder.DateCreated
' - Folder.DateLastAccessed
' - Folder.DateLastModified
' - Folder.Size
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Function GenerateFolderInformation(Folder)

	Dim S

	S = "Путь:" & TabStop & Folder.Path
	S = S & NewLine & "Имя:" & TabStop & Folder.Name
	S = S & NewLine & "Атрибуты:" & TabStop & ShowFileAttr(Folder)
	S = S & NewLine & "Создан:" & TabStop & Folder.DateCreated
	S = S & NewLine & "Доступен:" & TabStop & Folder.DateLastAccessed
	S = S & NewLine & "Изменён:" & TabStop & Folder.DateLastModified
	S = S & NewLine & "Размер:" & TabStop & Folder.Size & NewLine

	GenerateFolderInformation = S

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'
' GenerateAllFolderInformation
'
' Назначение:
'
' Генерация строки, описывающей текущее состояние папки
' и все файл и подкаталоги
'
' Демонстрируется следующее
'
' - Folder.Path
' - Folder.SubFolders
' - Folders.Count
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Function GenerateAllFolderInformation(Folder)

	Dim S
	Dim SubFolders
	Dim SubFolder
	Dim Files
	Dim File

	S = "Каталог:" & TabStop & Folder.Path & NewLine & NewLine

	Set Files = Folder.Files

	If 1 = Files.Count Then
		S = S & "Это 1 файл" & NewLine
	Else
		S = S & "Это " & Files.Count & " файлов" & NewLine
	End If

	If Files.Count <> 0 Then

		For Each File In Files
			S = S & GenerateFileInformation(File)
		Next

	End If

	Set SubFolders = Folder.SubFolders
```

```
        If 1 = SubFolders.Count Then
        S = S & NewLine & "Это 1 подкаталог" & NewLine & NewLine
        Else
                  S = S & NewLine & "Это " & SubFolders.Count & " подкаталогов" & NewLine & NewLine
        End If

        If SubFolders.Count <> 0 Then

                For Each SubFolder In SubFolders
                        S = S & GenerateFolderInformation(SubFolder)
                Next

                S = S & NewLine

                For Each SubFolder In SubFolders
                        S = S & GenerateAllFolderInformation(SubFolder)
                Next

        End If

        GenerateAllFolderInformation = S

End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'
' GenerateTestInformation
'
' Назначение:
'
' Генерация строки, описывающей текущее состояние папки C:\Test
' и всех файлов и подкаталогов
'
' Демонстрируется следующее
'
' - FileSystemObject.DriveExists
' - FileSystemObject.FolderExists
' - FileSystemObject.GetFolder
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Function GenerateTestInformation(FSO)

        Dim TestFolder
        Dim S

        If Not FSO.DriveExists(TestDrive) Then Exit Function
        If Not FSO.FolderExists(TestFilePath) Then Exit Function

        Set TestFolder = FSO.GetFolder(TestFilePath)

        GenerateTestInformation = GenerateAllFolderInformation(TestFolder)

End Function
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'
' DeleteTestDirectory
'
' Назначение:
'
' Очистка проверочной директории.
'
' Демонстрируется следующее
'
' - FileSystemObject.GetFolder
' - FileSystemObject.DeleteFile
' - FileSystemObject.DeleteFolder
' - Folder.Delete
' - File.Delete
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Sub DeleteTestDirectory(FSO)

        Dim TestFolder
        Dim SubFolder
        Dim File

        ' Два способа удаления файла:

        FSO.DeleteFile(TestFilePath & "\Beatles\OctopusGarden.txt")

        Set File = FSO.GetFile(TestFilePath & "\Beatles\BathroomWindow.txt")
        File.Delete

        ' Два способа удаления папки:

        FSO.DeleteFolder(TestFilePath & "\Beatles")

        FSO.DeleteFile(TestFilePath & "\ReadMe.txt")

        Set TestFolder = FSO.GetFolder(TestFilePath)
        TestFolder.Delete

End Sub
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'
' CreateLyrics
'
' Назначение:
'
' Создание пары текстовых файлов в папке.
'
' Демонстрируется следующее
'
' - FileSystemObject.CreateTextFile
' - TextStream.WriteLine
' - TextStream.Write
' - TextStream.WriteBlankLines
' - TextStream.Close
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Sub CreateLyrics(Folder)

        Dim TextStream

        Set TextStream = Folder.CreateTextFile("OctopusGarden.txt")

        TextStream.Write("Octopus' Garden ")
' Учтите, что это не добавляет символ конца строки в файл
        TextStream.WriteLine("(от Ринго Старр)")
        TextStream.WriteBlankLines(1)
TextStream.WriteLine("I'd like to be under the sea in an octopus' garden in the shade,")
TextStream.WriteLine("He'd let us in, knows where we've been -- in his octopus' garden in the shade.")
        TextStream.WriteBlankLines(2)

        TextStream.Close

        Set TextStream = Folder.CreateTextFile("BathroomWindow.txt")
             TextStream.WriteLine("She Came In Through The Bathroom Window (by Lennon/McCartney)")
        TextStream.WriteLine("")
        TextStream.WriteLine("She came in through the bathroom window protected by a silver spoon")
        TextStream.WriteLine("But now she sucks her thumb and wanders by the banks of her own lagoon")
        TextStream.WriteBlankLines(2)
        TextStream.Close

End Sub
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'
' GetLyrics
'
' Назначение:
'
' Отображение содержимого файлов.
'
' Демонстрируется следующее
'
' - FileSystemObject.OpenTextFile
' - FileSystemObject.GetFile
' - TextStream.ReadAll
' - TextStream.Close
' - File.OpenAsTextStream
' - TextStream.AtEndOfStream
' - TextStream.ReadLine
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Function GetLyrics(FSO)

        Dim TextStream
        Dim S
        Dim File

        ' Здесь несколько способов для открытия файла, и несколько способов для
        ' чтения данных из файла. По два способа для каждого действия:

Set TextStream = FSO.OpenTextFile(TestFilePath & "\Beatles\OctopusGarden.txt", OpenFileForReading)

        S = TextStream.ReadAll & NewLine & NewLine
        TextStream.Close

        Set File = FSO.GetFile(TestFilePath & "\Beatles\BathroomWindow.txt")
        Set TextStream = File.OpenAsTextStream(OpenFileForReading)
        Do      While Not TextStream.AtEndOfStream
                S = S & TextStream.ReadLine & NewLine
        Loop
        TextStream.Close

        GetLyrics = S

End Function
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'
' BuildTestDirectory
'
' Назначение:
'
' Создание иерархии каталогов для демонстрации FileSystemObject.
'
' Мы создадим иерархию в следующем порядке:
'
' C:\Test
' C:\Test\ReadMe.txt
' C:\Test\Beatles
' C:\Test\Beatles\OctopusGarden.txt
' C:\Test\Beatles\BathroomWindow.txt
'
'
' Демонстрируется следующее
'
' - FileSystemObject.DriveExists
' - FileSystemObject.FolderExists
' - FileSystemObject.CreateFolder
' - FileSystemObject.CreateTextFile
' - Folders.Add
' - Folder.CreateTextFile
' - TextStream.WriteLine
' - TextStream.Close
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Function BuildTestDirectory(FSO)

        Dim TestFolder
        Dim SubFolders
        Dim SubFolder
        Dim TextStream

        ' Выход из функции если диск не существует (a), или каталог уже существует (b)

        If Not FSO.DriveExists(TestDrive) Then
                BuildTestDirectory = False
                Exit Function
        End If

        If FSO.FolderExists(TestFilePath) Then
                BuildTestDirectory = False
                Exit Function
        End If

        Set TestFolder = FSO.CreateFolder(TestFilePath)

        Set TextStream = FSO.CreateTextFile(TestFilePath & "\ReadMe.txt")
        TextStream.WriteLine("Моя коллекция лирических песен")
        TextStream.Close

        Set SubFolders = TestFolder.SubFolders

        Set SubFolder = SubFolders.Add("Beatles")

        CreateLyrics SubFolder

        BuildTestDirectory = True

End Function
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'
' Основная процедура
'
' Сначала создаётся тестовая директория с несколькими подкаталогами и файлами.
' Затем она заполняется некоторой информацией о доступных дисках и о
' тестовой директории, а затем всё удаляется.
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Sub Main

        Dim FSO

        ' Установить глобальные данные.
        TabStop = Chr(9)
        NewLine = Chr(10)

        Set FSO = CreateObject("Scripting.FileSystemObject")

        If Not BuildTestDirectory(FSO) Then
        MsgBox "Тестовая директория уже существует или не может быть создана. Продолжение невозможно."
                Exit Sub
        End If

        MsgBox GenerateDriveInformation(FSO) & NewLine & NewLine

        MsgBox GenerateTestInformation(FSO) & NewLine & NewLine

        MsgBox GetLyrics(FSO) & NewLine & NewLine

        DeleteTestDirectory(FSO)

End Sub
```

5. СПРАВОЧНЫЕ МАТЕРИАЛЫ

5.1. Функции VBScript

5.1.1. Функции VBScript

В данном разделе приведён список функций, операторов и ключевых слов VBScript. Функции разбиты на категории.

Категория	Ключевые слова
Работа с массивами	Array Dim, Private, Public, ReDim IsArray Erase LBound, UBound
Назначение	Set
Комментарии	Для определения комментариев используется символ ' или слово Rem
Константы/Литералы	Empty Nothing Null True, False
Управление программой	Do...Loop For...Next For Each...Next If...Then...Else Select Case While...Wend With
Преобразования	Abs Asc, AscB, AscW Chr, ChrB, ChrW CBool, CByte CCur, CDate CDbl, CInt CLng, CSng, CStr DateSerial, DateValue Hex, Oct Fix, Int Sgn TimeSerial, TimeValue
Дата/время	Date, Time DateAdd, DateDiff, DatePart DateSerial, DateValue Day, Month, MonthName Weekday, WeekdayName, Year Hour, Minute, Second Now TimeSerial, TimeValue
Объявления	Class Const Dim, Private, Public, ReDim Function, Sub Property Get, Property Let, Property Set
Обработка ошибок	On Error Err

Категория	Ключевые слова
Выражения	Expressions Eval Execute RegExp Replace Test
Форматирование строк	FormatCurrency FormatDateTime FormatNumber FormatPercent
Ввод/вывод	InputBox LoadPicture MsgBox
Литералы	Empty False Nothing Null True
Математика	Atn, Cos, Sin, Tan Exp, Log, Sqr Randomize, Rnd
Разное	Eval Function Execute Statement RGB Function
Объекты	CreateObject Err Object GetObject RegExp
Операторы	Сложение (+), Вычитание (-) Возведение в степень (^) Арифметический модуль (Mod) Умножение (*), Деление (/) Целочисленное деление (\) Унарный минус (-) Объединение строк (&) Равно (=), Не равно (<>) Меньше (<), Меньше или равно (<=) Больше (>) Больше или равно (>=) Is And, Or, Xor Eqv, Imp
Опции	Option Explicit
Процедуры	Call Function, Sub Property Get, Property Let, Property Set
Округление	Abs Int, Fix, Round Sgn
Сценарий	ScriptEngine ScriptEngineBuildVersion ScriptEngineMajorVersion ScriptEngineMinorVersion

Категория	Ключевые слова
Строки	Asc, AscB, AscW Chr, ChrB, ChrW Filter, InStr, InStrB InStrRev Join Len, LenB LCase, UCase Left, LeftB Mid, MidB Right, RightB Replace Space Split StrComp String StrReverse LTrim, RTrim, Trim
Варианты	IsArray IsDate IsEmpty IsNull IsNumeric IsObject TypeName VarType

5.1.2. Функции VBA, которых нет в VBScript

В данном разделе приведён список функций Visual Basic for Applications (VBA), которых нет в VBScript. Этот раздел может оказаться полезен для тех, кто хорошо знаком с VBA.

Категория	Отсутствующая функция/ключевое слово
Работа с массивами	Option Base Объявление массивов с нижней границей не равной 0
Коллекции	Add, Count, Item, Remove Использование символа ! при работе с коллекциями (например, MyCollection!Foo)
Условная компиляция	#Const #If...Then...#Else
Управление программой	DoEvents GoSub...Return, GoTo On Error GoTo On...GoSub, On...GoTo Line numbers, Line labels
Преобразования	CVar, CVDate Str, Val
Типы данных	Все типы данных, кроме Variant Type...End Type
Дата/Время	Оператор Date, оператор Time
DDE	LinkExecute, LinkPoke, LinkRequest, LinkSend
Отладка	Debug.Print End, Stop
Объявление	Declare (for declaring DLLs) Optional ParamArray Static
Обработка ошибок	Erl Error Resume, Resume Next
Файловый ввод/вывод	Все традиционные операции файлового ввода/вывода Basic
Финансы	Все финансовые функции
Объекты	Clipboard Collection
Операторы	Like
Опции	Deftype Option Base Option Compare Option Private Module
Select Case	Выражения, содержащие ключевое слово **Is**, или какие-либо операторы сравнения. Выражения, содержащие диапазон значений, использующие ключевое слово **To**.
Строки	Строки с фиксированной длиной LSet, RSet Оператор Mid StrConv
Использование объектов	Использование символа ! при работе с коллекциями.

5.1.3. Функции VBScript, которых нет в VBA

В данном разделе приведён список функций VBScript, которых нет в Visual Basic for Applications (VBA). Этот раздел может оказаться полезен для тех, кто хорошо знаком с VBA.

Категория	Отсутствующая функция/ключевое слово
Объявления	Class
Разное	Eval Execute
Объекты	RegExp
Сценарий	ScriptEngine ScriptEngineBuildVersion ScriptEngineMajorVersion ScriptEngineMinorVersion

5.1.4. Функции Scripting Run-Time Library Reference Features

Категория	Функция/ключевое слово
Коллекции	Drives Files Folders
Управление файлами и системой	Dictionary Add, Exists, Items, Keys, Remove, RemoveAll Count, Item, Key Drive, File, Folder, Copy, Delete, Move, OpenAsTextStream Attributes, Count, DateCreated, DateLastAccessed, DateLastModified, Drive, ParentFolder, Name, Path, ShortName, ShortPath, Size AvailableSpace, DriveLetter, DriveType, FileSystem, FreeSpace, IsReady, RootFolder, SerialNumber, ShareName, TotalSize, VolumeName FileSystemObject BuildPath, CopyFile, CopyFolder, CreateFolder, CreateTextFile, DeleteFile, DeleteFolder, DriveExists, FileExists, FolderExists, GetAbsolutePathName, GetBaseName, GetDrive, GetDriveName, GetExtensionName, GetFile, GetFileName, GetFolder, GetParentFolderName, GetSpecialFolder, GetTempName, MoveFile, MoveFolder, OpenTextFile Drives TextStream Close, Read, ReadAll, ReadLine, Skip, SkipLine, Write, WriteBlankLines, WriteLine AtEndOfLine, AtEndOfStream, Column, Line

5.2. Список ключевых слов в алфавитном порядке

False
FileSystemObject Object
Filter функция
FirstIndex Property
Fix функция
For...Next Statement
For Each...Next Statement
FormatCurrency функция
FormatDateTime функция
FormatNumber функция
FormatPercent функция
Function Statement
GetObject функция
GetRef функция
Global функция
Hex функция
HelpContext Property
HelpFile Property
Hour функция
If...Then...Else Statement
IgnoreCase Property
Imp Operator
Initialize Event
InputBox функция
InStr функция
InStrRev функция
Int функция
Integer Division Operator (\)
Is Operator
IsArray функция
IsDate функция
IsEmpty функция
IsNull функция
IsNumeric функция
IsObject функция
Join функция
LBound функция
LCase функция
Left функция
Len функция
Length Property
LoadPicture функция
Log функция
LTrim функция
Match Object
Matches Collection
Mid функция
Minute функция
Miscellaneous Constants
Mod Operator
Month функция
MonthName функция
MsgBox Constants
MsgBox функция
Muliplication Operator (*)

Tan функция
Terminate Event
Test Method
Time функция
Timer функция
TimeSerial функция
TimeValue функция
Trim функция
Tristate Constants
True
TypeName функция
UBound функция
UCase функция
Value Property
VarType Constants
VarType функция
VBScript Constants
Weekday функция
WeekdayName функция
While...Wend Statement
With Statement
Xor Operator
Year функция

5.3. Константы

5.3.1. Константы VBScript

При работе с VBScript вы можете использовать константы в вашем коде. Константы предоставляют удобный способ для использования постоянных значений когда вам не требуется запоминать эти значения, а достаточно объявить константу и далее в программе использовать её осмысленное имя вместо маловразумительного значения. Использование констант делает ваш код более читабельным. Поэтому многие часто используемые значения уже определены в VBScript как константы и вам нет необходимости объявлять их в своём коде. Просто используйте имена этих констант вместо значений.

Ниже перечислены категории констант стандартных VBScript.

- **Цветовые константы**. Определяют восемь основных цветов, которые могут быть использованы в сценариях.
- **Константы времени и даты**. Определяют константы даты и времени, используемые в различных функциях.
- **Константы форматирования даты**. Определяют константы, используемые при форматировании даты и времени.
- **Константы разные**. Константы, которые не вошли в другие категории.
- **Константы MsgBox**. Константы, используемые функцией MsgBox.
- **Строковые константы**. Непечатаемые символы, используемые при операциях со строками.
- **Константы трёх состояний**. Константы, используемые при форматировании чисел.
- **Константы типов переменных**. Константы, определяющие подтипы данных типа Variant.

5.3.2. Цветовые константы

Так как эти константы встроены в VBScript, вам нет необходимости объявлять их перед использованием. Вы можете применять их в вашем коде для представления значений, описанных ниже в таблице:

Константа	Значение	Описание
vbBlack	&h00	Black (чёрный)
vbRed	&hFF	Red (красный)
vbGreen	&hFF00	Green (зелёный)
vbYellow	&hFFFF	Yellow (жёлтый)
vbBlue	&hFF0000	Blue (синий)
vbMagenta	&hFF00FF	Magenta (пурпурный)
vbCyan	&hFFFF00	Cyan (голубой)
vbWhite	&hFFFFFF	White (белый)

5.3.3. Константы сравнения

Так как эти константы встроены в VBScript, вам нет необходимости объявлять их перед использованием. Вы можете применять их в вашем коде для представления значений, описанных ниже в таблице:

Константа	Значение	Описание
vbBinaryCompare	0	Выполнить двоичное сравнение
vbTextCompare	1	Выполнить текстовое сравнение

5.3.4. Константы даты и времени

Так как эти константы встроены в VBScript, вам нет необходимости объявлять их перед использованием. Вы можете применять их в вашем коде для представления значений, описанных ниже в таблице:

Константа	Значение	Описание
vbSunday	1	Воскресенье
vbMonday	2	Понедельник
vbTuesday	3	Вторник
vbWednesday	4	Среда
vbThursday	5	Четверг
vbFriday	6	Пятница
vbSaturday	7	Суббота
vbUseSystem	0	Использует формат даты, содержащийся в настройках вашего компьютера
vbUseSystemDayOfWeek	0	Использует день недели, указанный в настройках вашей системы для первого дня недели
vbFirstJan1	1	Использует неделю, в которой содержится 1 января (по умолчанию)
vbFirstFourDays	2	Использует первую неделю, которая содержит не менее 4 дней нового года
vbFirstFullWeek	3	Использует первую полную неделю года

5.3.5. Константы форматирования даты

Так как эти константы встроены в VBScript, вам нет необходимости объявлять их перед использованием. Вы можете применять их в вашем коде для представления значений, описанных ниже в таблице:

Константа	Значение	Описание
vbGeneralDate	0	Отображает дату и/или время. Для вещественных чисел отображает дату и время. Если число не имеет дробной части, то отображает только дату. Если число не имеет целой части, то отображает только время. Отображение даты и времени определено настройками вашей системы.
vbLongDate	1	Отображает дату, используя длинный формат даты, указанный в региональных настройках вашей системы.
vbShortDate	2	Отображает дату, используя короткий формат даты, указанный в региональных настройках вашей системы.
vbLongTime	3	Отображает время, используя длинный формат времени, указанный в региональных настройках вашей системы.
vbShortTime	4	Отображает время, используя короткий формат времени, указанный в региональных настройках вашей системы.

5.3.6. Константы разные

Так как эти константы встроены в VBScript, вам нет необходимости объявлять их перед использованием. Вы можете применять их в вашем коде для представления значений, описанных ниже в таблице:

Константа	Значение	Описание
vbObjectError	-2147221504	Определённое пользователем количество ошибок должно быть больше этого значения, например Err.Raise Number = vbObjectError + 1000

5.3.7. Константы MsgBox

Следующие константы используются с функцией **MsgBox** для определения набора кнопок и значков, которые будут отображаться в окне сообщения и кнопки по умолчанию. Кроме этого может быть указана модальность **MsgBox**. Так как эти константы встроены в VBScript, вам нет необходимости объявлять их перед использованием. Вы можете применять их в вашем коде для представления значений, описанных ниже в таблице:

Константа	Значение	Описание
vbOKOnly	0	Отображать только кнопку OK
vbOKCancel	1	Отображать кнопки OK и Cancel (Отмена)
vbAbortRetryIgnore	2	Отображать кнопки Abort (Прервать), Retry (Повтор) и Ignore (Игнорировать)
vbYesNoCancel	3	Отображать кнопки Yes (Да), No (Нет) и Cancel (Отмена)
vbYesNo	4	Отображать кнопки Yes (Да) и No (Нет)
vbRetryCancel	5	Отображать кнопки Retry (Повтор) и Cancel (Отмена)
vbCritical	16	Отображать значок «Критическое сообщение»
vbQuestion	32	Отображать значок «Предупредительный запрос»
vbExclamation	48	Отображать значок «Предупреждение»
vbInformation	64	Отображать значок «Информационное сообщение»
vbDefaultButton1	0	По умолчанию фокус на первой кнопке
vbDefaultButton2	256	По умолчанию фокус на второй кнопке
vbDefaultButton3	512	По умолчанию фокус на третьей кнопке
vbDefaultButton4	768	По умолчанию фокус на четвёртой кнопке
vbApplicationModal	0	Модальное приложение. Пользователь должен ответить на вопрос, поставленный в окне сообщения, прежде чем продолжить работу в текущем приложении.
vbSystemModal	4096	Модальная система. В системах Win16 все приложения подвешены до тех пор, пока пользователь не работает с окном сообщения. В системах Win32 эта константа предоставляет приложению модальное окно сообщения, которое всегда находится поверх других программ, которые вы можете выполнять.

Следующие константы используются с функцией **MsgBox** для определения кнопки, которую нажал пользователь. Эти константы доступны только когда ваш проект явно ссылается на соответствующую библиотеку типов, содержащую определения этих констант. Для VBScript вы должны явно определить эти константы в вашем коде.

Константа	Значение	Описание
vbOK	1	Была нажата кнопка OK
vbCancel	2	Была нажата кнопка Cancel (Отмена)
vbAbort	3	Была нажата кнопка Abort (Прервать)
vbRetry	4	Была нажата кнопка Retry (Повтор)
vbIgnore	5	Была нажата кнопка Ignore (Игнорировать)
vbYes	6	Была нажата кнопка Yes (Да)
vbNo	7	Никакая кнопка не была нажата

5.3.8. Строковые константы

Так как эти константы встроены в VBScript, вам нет необходимости объявлять их перед использованием. Вы можете применять их в вашем коде для представления значений, описанных ниже в таблице:

Константа	Значение	Описание
vbCr	Chr(13)	Возврат каретки
vbCrLf	Chr(13) & Chr(10)	Возврат каретки и переход на новую строку
vbFormFeed	Chr(12)	Вставка, не используется в Microsoft Windows
vbLf	Chr(10)	Переход на новую строку
vbNewLine	Chr(13) & Chr(10) или Chr(10)	Символ новой строки, не зависящий от платформы, то есть тот из двух, который соответствует платформе
vbNullChar	Chr(0)	Символ, имеющий значение 0
vbNullString	Пустая строка	Строка с нулевой длиной (“”), используется для вызывающих внешних процедур
vbTab	Chr(9)	Горизонтальная табуляция
vbVerticalTab	Chr(11)	Вертикальная табуляция, не используется в Microsoft Windows

5.3.9. Константы трёх состояний

Так как эти константы встроены в VBScript, вам нет необходимости объявлять их перед использованием. Вы можете применять их в вашем коде для представления значений, описанных ниже в таблице:

Константа	Значение	Описание
vbUseDefault	-2	Использует по умолчанию региональные настройки компьютера
vbTrue	-1	True (Истина)
vbFalse	0	False (Ложь)

5.3.10. Константы типов переменных

Эти константы доступны только когда ваш проект явно ссылается на соответствующую библиотеку типов, содержащую определения этих констант. Для VBScript вы должны явно определить эти константы в вашем коде.

Константа	Значение	Описание
vbEmpty	0	Не инициализирован (по умолчанию)
vbNull	1	Не содержит данных
vbInteger	2	Integer
vbLong	3	Long
vbSingle	4	Single
vbDouble	5	Double
vbCurrency	6	Currency
vbDate	7	Date
vbString	8	String
vbObject	9	Object
vbError	10	Error
vbBoolean	11	Boolean
vbVariant	12	Variant (используется только для массивов вариантов)
vbDataObject	13	Data access object
vbDecimal	14	Decimal
vbByte	17	Byte
vbArray	8192	Array

5.4. События

5.4.1. Событие Initialize

Описание

Событие происходит при создании экземпляра связанного класса.

Синтаксис

```
Private Sub ИмяКласса_Initialize()
  операторы
End Sub
```

Событие **Initialize** имеет следующие составляющие:

Элемент	Описание
ИмяКласса	Обязательный элемент. Имя класса, определённое с помощью оператора **Class**.
Операторы	Не обязательный параметр. Код, который выполняется при инициализации класса.

5.4.2. Событие Terminate

Описание

Событие происходит при завершении экземпляра связанного класса.

Синтаксис

```
Private Sub ИмяКласса_Terminate()
  операторы
End Sub
```

Событие **Terminate** имеет следующие составляющие:

Элемент	Описание
ИмяКласса	Обязательный элемент. Имя класса, определённое с помощью оператора **Class**.
Операторы	Не обязательный параметр. Код, который выполняется при завершении класса.

5.5. Функции

Abs Function
Array Function
Asc Function
Atn Function
CBool Function
CByte Function
CCur Function
CDate Function
CDbl Function
Chr Function
CInt Function
CLng Function
Cos Function
CreateObject Function
CSng Function
CStr Function
Date Function
DateAddFunction
DateDiff Function
DatePart Function
DateSerial Function
DateValue Function
Day Function
Eval Function
Exp Function
Filter Function
Fix Function
FormatCurrency Function
FormatDateTime Function
FormatNumber Function
FormatPercent Function
GetObject Function
GetRef Function
Hex Function
Hour Function
InputBox Function
InStr Function
InStrRev Function
Int Function
IsArray Function
IsDate Function
IsEmpty Function
IsNull Function
IsNumeric Function
IsObject Function
Join Function
LBound Function
LCase Function
Left Function
Len Function
LoadPicture Function
Log Function
LTrim Function

Mid Function
Minute Function
Month Function
MonthName Function
MsgBox Function
Now Function
Oct Function
Replace Function
RGB Function
Right Function
Rnd Function
Round Function
RTrim Function
ScriptEngine Function
ScriptEngineBuildVersion Function
ScriptEngineMajorVersion Function
ScriptEngineMinorVersion Function
Second Function
Sgn Function
Sin Function
Space Function
Split Function
Sqr Function
StrComp Function
String Function
StrReverse Function
Tan Function
Time Function
Timer Function
TimeSerial Function
TimeValue Function
Trim Function
TypeName Function
UBound Function
UCase Function
VarType Function
Weekday Function
WeekdayName Function
Year Function

5.6. Методы

Clear Method
Execute Method
Raise Method
Replace Method
Test Method

5.7. Объекты

5.7.1. Объект Class

Объект предоставляет доступ к событиям класса (см. раздел «5.4. События»). Объект создаётся с помощью оператора **Class**.

Вы не можете явно объявить переменную типа **Class**. В контексте VBScript объект Class означает любой объект, определённый с использованием ключевого слова Class. Однажды создав класс с использованием ключевого слова Class, вы можете создавать экземпляры класса, используя следующий синтаксис:

```
Dim X
Set X = New ИмяКласса
```

Поскольку VBScript использует позднее связывание, вы не можете использовать ни один из приведённых ниже примеров синтаксиса:

```
Dim X as New ИмяКласса
```

или

```
Dim X
X = New ИмяКласса
```

или

```
Set X = New Scripting.FileSystemObject
```

5.7.2. Объект Dictionary

Объект предназначен для создания, хранения и чтения списка пар ключ-значение. Объект **Dictionary** аналогичен связанному массиву в PERL. Значения, которые могут быть любыми типами данных, сохраняются в массив, каждый элемент которого связан с уникальным ключом. Этот ключ используется для доступа к отдельным элементам и обычно ключ является целым числом или строкой, но в принципе может иметь и другой тип, за исключением массива.

Следующий пример показывает, как создать объект **Dictionary**:

```
Dim d                  ' Создать переменную
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Ауди"      ' Добавить несколько ключей и элементов
d.Add "b", "БМВ"
d.Add "m", "Мерседес"

MsgBox d.Item("m")
```

5.7.2.1. Свойства объекта Dictionary

Свойство CompareMode

С помощью этого свойства можно установить или получить режим сравнения ключевых строк в объекте **Dictionary**. Синтаксис:

```
Объект.CompareMode[ = Сравнение]
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Dictionary**.
Сравнение	Не обязательный параметр. Если имеется, то представляет режим сравнения, используемый функциями, такими как **StrComp**.

Аргумент **Сравнение** может принимать следующие значения:

Константа	Значение	Описание
vbBinaryCompare	0	Выполняет двоичное сравнение
vbTextCompare	1	Выполняет текстовое сравнение

ПРИМЕЧАНИЯ

Значения более 2 могут быть использованы, если при сравнении применяется Locale IDs (LCID). Если изменить режим сравнения объекта **Dictionary**, который уже содержит данные, то произойдёт ошибка.

Свойство **CompareMode** использует те же значения, что и аргумент **Сравнение** функции **StrComp**.

В следующем примере используется свойство **CompareMode**:

```
Dim d                   ' Создать переменную
Set d = CreateObject("Scripting.Dictionary")
d.CompareMode = vbTextCompare
d.Add "a", "Ауди"       ' Добавить несколько ключей и элементов
d.Add "b", "БМВ"
d.Add "m", "Мерседес"
d.Add "M", "МАЗ"        ' Метод Add вызовет ошибку, так как в объекте
                        ' уже имеется элемент с таким ключом
```

Если в данном примере выражение **d.CompareMode = vbTextCompare** заменить на **d.CompareMode = vbBinaryCompare**, то ошибки не произойдёт, так как в этом случае символы **m** и **M** будут расцениваться как разные символы.

Свойство Count

С помощью этого свойства можно получить количество элементов в коллекции или в объекте **Dictionary**. Синтаксис:

```
Объект.Count
```

В следующем примере используется свойство **Count**:

```
Function ShowKeys
  Dim a, d, i, s           ' Создать переменные
  Set d = CreateObject("Scripting.Dictionary")
  d.Add "a", "Ауди"        ' Добавить несколько ключей и элементов
  d.Add "b", "БМВ"
  d.Add "m", "Мерседес"
  a = d.Keys               ' Получить ключи
  For i = 0 To d.Count -1 ' Перебрать массив
    s = s & a(i) & ", "    ' Создать результирующую строку
  Next
  ShowKeys = s
End Function
```

Свойство Item

С помощью этого свойства можно установить или получить **Элемент** в объекте **Dictionary**. Элемент определяется по параметру **Ключ**. Синтаксис:

```
Объект.Item(Ключ) [= НовыйЭлемент]
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Dictionary**.
Ключ	Обязательный элемент. Ключ, связанный с элементом, который должен быть предварительно найден или добавлен.
НовыйЭлемент	Не обязательный параметр. Используется только для объекта **Dictionary**. Не применяется для коллекций. Если имеется, то **НовыйЭлемент** – это новое значение, связанное с указанным ключом.

ПРИМЕЧАНИЕ
Если в массиве не найден элемент с ключом **Ключ** при смене элемента, то создаётся новый **Ключ**, с которым связывается новый элемент, значение которого указано в параметре **НовыйЭлемент**. Если **Ключ** не найден при попытке получить существующий элемент, то создаётся новый **Ключ**, который ссылается на пустой элемент.

В следующем примере используется свойство **Item**:

```
Dim d, s                  ' Создать переменные
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Ауди"         ' Добавить несколько ключей и элементов
d.Add "b", "БМВ"
d.Add "m", "Мерседес"

s = d.Item("x")           ' Пытаемся получить несуществующий элемент
d.Item("z") = "КаМАЗ"     ' Создать элемент и присвоить значение
```

Свойство Key

С помощью этого свойства можно установить **Ключ** в объекте **Dictionary**. Синтаксис:

```
Объект.Key(Ключ) = НовыйКлюч
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Dictionary**.
Ключ	Обязательный элемент. Ключ, значение которого требуется изменить.
НовыйКлюч	Обязательный элемент. Новое значение, которое заменяет указанный **Ключ**.

ПРИМЕЧАНИЕ
Если указанный **Ключ** не найден, то происходит ошибка.

В следующем примере используется свойство **Key**:

```
Function DicDemo
  Dim d                    ' Создать переменные
  Set d = CreateObject("Scripting.Dictionary")
  d.Add "a", "Ауди"        ' Добавить несколько ключей и элементов
  d.Add "b", "БМВ"
  d.Add "m", "Мерседес"
  d.Key("m") = "c"         ' Заменить ключ "m" на "c"
  DicDemo = d.Item("c")    ' Вернуть элемент, связанный с изменённым ключом
End Function
```

5.7.2.2. Методы объекта Dictionary

Метод Add

Добавляет в объект **Dictionary** пару ключ/элемент. Синтаксис:

```
Объект.Add Ключ, Элемент
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Dictionary**.
Ключ	Обязательный элемент. **Ключ**, который будет связан с создаваемым **элементом**.
Элемент	Обязательный элемент. **Элемент**, который будет связан с создаваемым **ключом**.

Если **Ключ** уже существует, то произойдёт ошибка.

В следующем примере используется метод **Add**:

```
  Dim d                    ' Создать переменные
  Set d = CreateObject("Scripting.Dictionary")
  d.Add "a", "Ауди"        ' Добавить несколько ключей и элементов
  d.Add "b", "БМВ"
  d.Add "m", "Мерседес"
```

Метод Exists

Возвращает TRUE, если указанный ключ существует в объекте **Dictionary**, иначе возвращает FALSE. Синтаксис:

```
Объект.Exists(Ключ)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Dictionary**.
Ключ	Обязательный элемент. **Ключ**, который требуется найти в объекте **Dictionary**.

В следующем примере используется метод **Exists**:

```
Function KeyExistsDemo
  Dim d, msg              ' Создать переменные
  Set d = CreateObject("Scripting.Dictionary")
  d.Add "a", "Ауди"       ' Добавить несколько ключей и элементов
  d.Add "b", "БМВ"
  d.Add "m", "Мерседес"
  If d.Exists("m") Then
    msg = "Ключ m существует"
  Else
    msg = "Ключ m не найден"
  End If
  KeyExistsDemo = msg
End Function
```

Метод Items

Возвращает массив, содержащий все элементы объекта **Dictionary**. Синтаксис:

```
Объект.Items
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Dictionary**.

В следующем примере используется метод **Items**:

```
Function DicDemo
  Dim a, d, i, s          ' Создать переменные
  Set d = CreateObject("Scripting.Dictionary")
  d.Add "a", "Ауди"       ' Добавить несколько ключей и элементов
  d.Add "b", "БМВ"
  d.Add "m", "Мерседес"
  a = d.Items             ' Получить все элементы
  For i = 0 To d.Count - 1 ' Перебрать массив
    s = s & a(i) & vbCrLf  ' Создать результирующую строку
  Next
  DicDemo = s
End Function
```

Метод Keys

Возвращает массив, содержащий все существующие ключи объекта **Dictionary**. Синтаксис:

```
Объект.Keys
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Dictionary**.

В следующем примере используется метод **Keys**:

```
Function DicDemo
  Dim a, d, i, s           ' Создать переменные
  Set d = CreateObject("Scripting.Dictionary")
  d.Add "a", "Ауди"        ' Добавить несколько ключей и элементов
  d.Add "b", "БМВ"
  d.Add "m", "Мерседес"
  a = d.Keys               ' Получить все ключи
  For i = 0 To d.Count - 1 ' Перебрать массив
    s = s & a(i) & vbCrLf  ' Создать результирующую строку
  Next
  DicDemo = s
End Function
```

Метод Remove

Удаляет из объекта **Dictionary** пару ключ-элемент. Синтаксис:

```
Объект.Remove(Ключ)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Dictionary**.
Ключ	Обязательный элемент. **Ключ**, связанный с ключом элемента, который (вместе с ключом) будет удалён из объекта **Dictionary**.

Если указанный ключ не существует, то происходит ошибка.

В следующем примере используется метод **Remove**:

```
Dim a, d                  ' Создать переменные
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Ауди"         ' Добавить несколько ключей и элементов
d.Add "b", "БМВ"
d.Add "m", "Мерседес"
...
d.Remove("b")             ' Удалить вторую пару ключ/элемент
```

Метод RemoveAll

Удаляет из объекта **Dictionary** все ключи и связанные с ними элементы. Синтаксис:

```
Объект.RemoveAll
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Dictionary**.

В следующем примере используется метод **RemoveAll**:

```
Dim a, d                    ' Создать переменные
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Ауди"           ' Добавить несколько ключей и элементов
d.Add "b", "БМВ"
d.Add "m", "Мерседес"
...
d.RemoveAll                 ' Удалить все элементы
```

5.7.3. Объект Err

Объект содержит информацию об ошибках, которые произошли во время выполнения сценария. Для генерации и сброса ошибок используются методы **Raise** и **Clear**.

Объект **Err** – это важный глобальный объект. Вам нет необходимости создавать экземпляры этого объекта в вашем коде. Свойства этого объекта устанавливаются генератором ошибок.

По умолчанию объект **Err** имеет свойство **Number**. Свойство **Err.Number** содержит целое число и может быть использовано объектом автоматизации для получения значения SCODE.

Когда происходит ошибка во время выполнения сценария, свойства объекта **Err** заполняются информацией, которая однозначно определяет ошибку и информацией, которая будет использована для обработки ошибки. Ошибку можно также генерировать. Для генерации ошибки во время выполнения используйте метод **Raise**.

Свойства объекта **Err** сбрасываются в ноль или в пустую строку (“”) после выполнения оператора **On Error Resume Next**. Метод **Clear** можно использовать для явного сброса ошибки.

В следующем примере показано использование метода **Err**:

```
On Error Resume Next
Err.Raise 6  ' Вызвать ошибку переполнения
MsgBox ("Error # " & CStr(Err.Number) & " " & Err.Description)
Err.Clear    ' Сбросить ошибку
```

5.7.3.1. Свойства объекта Err

Свойство Description

Через это свойство можно получить или установить строку с описанием ошибки. Синтаксис:

```
Объект.Description [= ОписаниеОшибки]
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Err**.
ОписаниеОшибки	Необязательный элемент. Строковое выражение, содержащее описание ошибки.

В следующем примере используется свойство **Description**:

```
On Error Resume Next
Err.Description = "Наше описание ошибки переполнения"
Err.Raise 6  ' Вызвать ошибку переполнения
MsgBox ("Error # " & CStr(Err.Number) & " " & Err.Description)
Err.Clear    ' Сбросить ошибку
```

Свойство HelpContext

Через это свойство можно получить или установить идентификатор раздела (ID) справочного файла типа HLP. Синтаксис:

```
Объект.HelpContext [= ИдентификаторРаздела]
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Err**.
ИдентификаторРаздела	Необязательный элемент. Правильный идентификатор раздела справки в справочном файле. Если параметр не задан, то отображается раздел с идентификатором 0.

ПРИМЕЧАНИЕ
Если имя справочного файла указано в свойстве **Helpfile** объекта **Err**, то свойство **HelpContext** используется для автоматического отображения раздела справки, идентификатор которого указан в этом свойстве. Если оба свойства (**Helpfile** и **HelpContext**) пустые, то проверяется значение свойства **Number**. Если там находится код ошибки VBScript, то используется идентификатор раздела справки для описания этой ошибки. Если свойство **Number** не содержит ошибку VBScript, то на экране отображается содержимое справочного файла VBScript.

В качестве справочного файла можно использовать только файлы HLP. Справочные файлы другого типа (например, CHM) VBScript не воспринимает.

Идентификаторы разделов можно объявить как константы (см. пример сценария ниже) либо сразу использовать числовые значения при установке свойства **HelpContext** (например, Err.HelpContext = 2). Если распознать раздел по имени не удаётся (например, вы не определили константу или указали несуществующий идентификатор), то отображается раздел с идентификатором 0.

В следующем примере используется свойство **HelpContext**:

```
const IDH_Test = 0
const IDH_2 = 1
const IDLast = 2

On Error Resume Next
Dim Msg
Err.Clear
Err.Raise 6    ' Вызвать ошибку переполнения
Err.Helpfile = "HELP-TEST.HLP"
Err.HelpContext = IDH_Test
If Err.Number <> 0 Then
   Msg = "Нажмите F1 или кнопку СПРАВКА для просмотра " & Err.Helpfile & _
   " по теме, на которую ссылается HelpContext: " & Err.HelpContext
   MsgBox Msg, , "ошибка: " & Err.Description, Err.Helpfile,
Err.HelpContext
End If
```

Свойство HelpFile

Через это свойство можно получить или установить полный путь к справочному файлу типа HLP. Синтаксис:

```
Объект.HelpFile [= ПутьКФайлу]
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Err**.
ПутьКФайлу	Необязательный элемент. Полный путь к файлу справки.

ПРИМЕЧАНИЕ
Справочный файл указанный в свойстве **Helpfile**, автоматически вызывается когда пользователь щёлкает кнопку СПРАВКА или нажимает клавишу F1 в диалоговом окне сообщения об ошибке. Если свойство **HelpContext** содержит правильный идентификатор раздела справки указанного файла, то автоматически отображается этот раздел. Если файл справки в свойстве **Helpfile** не указан, то отображается файл справки VBScript.

В следующем примере используется свойство **Helpfile**:

```
const IDH_Test = 0
const IDH_2 = 1
const IDLast = 2

On Error Resume Next
Dim Msg
Err.Clear
Err.Raise 6    ' Вызвать ошибку переполнения
Err.Helpfile = "HELP-TEST.HLP"
Err.HelpContext = IDH_Test
If Err.Number <> 0 Then
   Msg = "Нажмите F1 или кнопку СПРАВКА для просмотра " & Err.Helpfile & _
   " по теме, на которую ссылается HelpContext: " & Err.HelpContext
   MsgBox Msg, , "ошибка: " & Err.Description, Err.Helpfile, Err.HelpContext
End If
```

Свойство Number

Через это свойство можно получить или установить числовой код ошибки. **Number** – это свойство объекта **Err** по умолчанию. Синтаксис:

```
Объект.Number [= НомерОшибки]
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Err**.
НомерОшибки	Целочисленное значение кода ошибки VBScript или значение ошибки SCODE.

ПРИМЕЧАНИЕ
Если возвращается определённый пользователем код ошибки из объекта автоматизации, установите свойство **Err.Number** путём добавления к константе **vbObjectError** выбранного вами числа в качестве кода ошибки.

В следующем примере используется свойство **Number**:

```
On Error Resume Next
Err.Raise vbObjectError + 1, "SomeObject" ' Вызвать ошибку #1 некоторого объекта
MsgBox ("Ошибка # " & CStr(Err.Number) & " " & Err.Description)
Err.Clear                                 ' Сбросить ошибку
```

Свойство Source

Через это свойство можно получить или установить имя объекта, который генерирует ошибку. Синтаксис:

```
Объект.Source [= СтроковоеВыражение]
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Err**.
СтроковоеВыражение	Строковое выражение, которое представляет приложение, генерирующее ошибку.

ПРИМЕЧАНИЯ

Свойство **Source** определяет строковое выражение, которое обычно содержит имя класса или программный идентификатор (ID) объекта, вызывающего ошибку. Использование свойства **Source** может предоставить вашим пользователям информацию в том случае, когда в вашем коде невозможно обработать ошибку, сгенерированную связанным объектом. Например, если вы подключились к **Microsoft Excel**, которая сгенерировала ошибку «Деление на ноль», то **Microsoft Excel** установит свой код ошибки в **Err.Number**, а в свойстве **Source** будет «Excel.Application». Учтите, что если ошибка была сгенерирована другим объектом, вызванным из **Microsoft Excel**, то **Excel** перехватит ошибку и установит свой код ошибки «Деление на ноль» в свойстве **Err.Number**.

Свойство **Source** всегда содержит имя объекта, который непосредственно генерирует ошибку. В вашем коде вы можете обработать эту ошибку в соответствии с документацией по ошибкам данного объекта. Если ваш обработчик ошибок отсутствует, вы можете использовать информацию объекта **Err** для описания ошибки и предоставить её пользователю.

Если ошибка генерируется в коде, то свойство **Source** содержит программный идентификатор приложения.

В следующем примере используется свойство **Source**:

```
On Error Resume Next
Err.Raise vbObjectError + 1, "SomeObject"' Вызвать ошибку #1 некоторого объекта
MsgBox ("Ошибка # " & CStr(Err.Number) & " " & Err.Description & Err.Source)
Err.Clear                              ' Сбросить ошибку
```

В приведённом выше примере свойство **Source** будет содержать строку «SomeObject». А в следующем примере свойство **Source** будет содержать строку «Microsoft VBScript».

```
On Error Resume Next
Err.Raise 6    ' Вызвать ошибку переполнения
MsgBox ("Ошибка # " & CStr(Err.Number) & " " & Err.Description & "/" & Err.Source)
Err.Clear               ' Сбросить ошибку
```

5.7.3.2. Методы объекта Err

Метод Clear

Данный метод очищает все свойства объекта **Err**. Синтаксис:

```
Объект.Clear
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Err**.

ПРИМЕЧАНИЕ
Используйте метод **Clear** для явного сброса свойств объекта **Err** после обработки ошибки. Это может быть необходимо, если вы используете отложенную обработку ошибки с **On Error Resume Next**. VBScript вызывает метод Clear автоматически каждый раз, когда выполняется один из следующих операторов:

- On Error Resume Next
- Exit Sub
- Exit Function

В следующем примере используется метод **Clear**:

```
On Error Resume Next
On Error Resume Next
Err.Raise 6         ' Вызвать ошибку переполнения
MsgBox ("Ошибка # " & CStr(Err.Number) & " " & Err.Description)
Err.Clear     ' Сбросить ошибку
```

Метод Raise

Данный метод генерирует ошибку времени выполнения. Синтаксис:

```
Объект.Raise(Номер, Источник, Описание, ФайлСправки, РазделСправки)
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Err**.
Номер	Целое число подтипа Long, которое идентифицирует природу ошибки. Ошибки VBScript (как предопределённые ошибки VBScript, так и ошибки определённые пользователем) находятся в диапазоне 0…65535.
Источник	Строковое выражение, определяющее имя объекта или приложения, которое непосредственно сгенерировало ошибку. Когда устанавливаете это свойство для объекта автоматизации, используёте формат project.class. Если этот параметр не указан, то используется программный идентификатор текущего проекта VBScript.
Описание	Строковое выражение, описывающее ошибку. Если не указано, то анализируется значение, указанное в параметре **Номер**. Если это значение может быть распознано как код ошибки времени выполнения VBScript, то строка, предоставленная от VBScript используется в этом параметре.
ФайлСправки	Полный путь к файлу справки, в котором может содержаться описание ошибки. Если не указано, то VBScript используется полный путь к файлу справки VBScript.
РазделСправки	Идентификатор раздела справки, который находится в файле справки, указанном в параметре **ФайлСправки**. Если не указан, то идентификатор определяется по значению, содержащемуся в параметре **Номер** и открывается файл справки VBScript с соответствующим разделом, если таковой существует.

ПРИМЕЧАНИЕ

Все параметры, кроме параметра **Номер**, не являются обязательными. Если вы используете метод **Raise** без указания некоторых параметров, а свойства объекта **Err** содержат значения, которые не были созданы в методе **Raise**, то будут использоваться свойства метода **Err**.

Если вы создаёте свою собственную ошибку для объекта автоматизации, то код ошибки в параметре **Номер** должен определяться путём прибавления вашего кода ошибки к константе **vbObjectError**. Например, для генерации ошибки 1050, задайте параметр **Номер** как значение vbObjectError + 1050.

В следующем примере используется метод **Raise**:

```
On Error Resume Next
On Error Resume Next
Err.Raise 6         ' Вызвать ошибку переполнения
MsgBox ("Ошибка # " & CStr(Err.Number) & " " & Err.Description)
Err.Clear     ' Сбросить ошибку
```

5.7.4. Объект FileSystemObject

Объект предоставляет доступ к файловой системе компьютера.

Следующий пример иллюстрирует, как использовать объект FileSystemObject для получения объекта TextStream, с помощью которого можно читать или записывать файлы:

```
Dim fso, MyFile
Set fso = CreateObject("Scripting.FileSystemObject")
Set MyFile = fso.CreateTextFile("testfile.txt", True)
MyFile.WriteLine("Это проверка")
MyFile.Close
```

Здесь функция **CreateObject** возвращает ссылку на объект **FileSystemObject** (fso). Метод **CreateTextFile** создаёт файл как объект **TextStream**, а метод **WriteLine** записывает строку текста в созданный текстовый файл. Метод **Close** очищает буфер и закрывает файл.

5.7.4.1. Свойства объекта FileSystemObject

Свойство Drives

Через это свойство можно получить коллекцию **Drives** (диски), содержащую все объекты **Drive**, доступные на локальном компьютере. Синтаксис:

```
Объект.Drives
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.

ПРИМЕЧАНИЕ
Все съёмные дисководы, в которые не вставлены диски, а также сетевые диски содержатся в коллекции **Drives**.

Вы можете перебрать элементы коллекции **Drives**, используя конструкцию **For Each...Next** как показано в следующем примере:

```
Function ShowDriveList
  Dim fso, d, dc, s, n
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set dc = fso.Drives
  For Each d in dc
    n = ""
    s = s & d.DriveLetter & " - "
    If d.DriveType = 3 Then
      n = d.ShareName
    ElseIf d.IsReady Then
      n = d.VolumeName
    End If
    s = s & n & vbCrLf
  Next
  ShowDriveList = s
End Function
```

5.7.4.2. Методы объекта FileSystemObject

Метод BuildPath

Метод добавляет имя к существующему пути. Синтаксис:

```
Объект.BuildPath(Путь, Имя)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
Путь	Обязательный элемент. Существующий путь, к которому добавляется **Имя**. Путь может быть абсолютным или относительным (без указания текущей папки).
Имя	Обязательный элемент. Имя, которое будет добавлено к существующему пути, указанному в параметре **Путь**.

ПРИМЕЧАНИЕ

Метод **BuildPath** вставляет в создаваемый путь разделитель между существующим путём и именем только в случае необходимости.

В следующем примере показано использование метода **BuildPath**:

```
Function GetBuildPath(path)
  Dim fso, newpath
  Set fso = CreateObject("Scripting.FileSystemObject")
  newpath = fso.BuildPath(path, "Подкаталог")
  GetBuildPath = newpath
End Function
```

Метод CopyFile

Метод копирует один или несколько файлов из одного места в другое. Синтаксис:

```
Объект.CopyFile Источник, Приёмник[, Перезапись]
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
Источник	Обязательный элемент. Строка символов, определяющая путь для одного или нескольких файлов, которые нужно копировать. Может содержать шаблон (маску), если нужно копировать несколько файлов.
Приёмник	Обязательный элемент. Строка символов, определяющая путь, куда будут скопированы файлы, указанные в параметре **Источник**. Не может содержать шаблоны (подстановочные символы, такие как *).
Перезапись	Не обязательный параметр. Логическое значение, которое определяет поведение метода, если файл, указанный в параметре **Приёмник**, уже существует. Если равно True, то файлы будут перезаписаны, если равно False, то копирование не выполнится. По умолчанию равно True. Учтите, что метод **CopyFile** завершится неудачно, если параметр **Перезапись** равен True, а файл, указанный в параметре **Приёмник** существует и имеет атрибут «только чтение».

ПРИМЕЧАНИЕ

Подстановочные символы могут использоваться только в последней части параметра **Источник**. Например:

```
FileSystemObject.CopyFile "c:\mydocuments\letters\*.doc", "c:\tempfolder\"
```

Но вы не можете использовать такой шаблон:

```
FileSystemObject.CopyFile "c:\mydocuments\*\R1???97.xls", "c:\tempfolder"
```

Если **Источник** содержит подстановочные символы или **Приёмник** заканчивается символом разделения пути (\), это подразумевает, что **Приёмник** – это существующая папка, в которую будут копироваться файлы, соответствующие шаблону. Иначе предполагается, что **Приёмник** – это имя файла, который будет создан и в который будет скопирован файл-источник. В любом случае могут произойти следующие события при копировании файла:

- Если **Приёмник** не существует, то файл копируется в **Источник**. То есть файлы будут скопированы в ту же папку, на которую указывает **Источник**.
- Если **Приёмник** – это существующий файл, а флаг **Перезапись** равен False, то происходит ошибка. Иначе (если перезапись разрешена), выполняется попытка перезаписать существующий файл файлом, указанным в параметре **Источник**.
- Если **Приёмник** – это не существующая директория, то происходит ошибка. То есть если в параметре **Приёмник** указан путь к директории с символом \ в конце, то эта директория должна существовать.
- Если **Приёмник** – это имя файла без расширения, которое совпадает с именем существующей директории, то происходит ошибка. Иначе создаётся файл без расширения.

Ошибка также случается, если **Источник** использует подстановочные символы, которым не соответствуют никакие файлы. Метод **CopyFile** прекращает работу при обнаружении первой ошибки. Все выполненные перед ошибкой операции не отменяются.

Метод CopyFolder

Метод рекурсивно копирует папку из одного места в другое. Синтаксис:

```
Объект.CopyFolder Источник, Приёмник[, Перезапись]
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
Источник	Обязательный элемент. Строка символов, определяющая путь для одного или нескольких каталогов, которые нужно копировать. Может содержать шаблон (маску), если нужно копировать несколько каталогов.
Приёмник	Обязательный элемент. Строка символов, определяющая путь, куда будут скопированы каталоги, указанные в параметре **Источник**. Не может содержать шаблоны (подстановочные символы, такие как *).
Перезапись	Не обязательный параметр. Логическое значение, которое определяет поведение метода, если каталог, указанный в параметре **Приёмник**, уже существует. Если равно True, то каталоги будут перезаписаны, если равно False, то копирование не выполнится. По умолчанию равно True.

ПРИМЕЧАНИЕ

Подстановочные символы могут использоваться только в последней части параметра **Источник**. Например:

```
FileSystemObject.CopyFolder "c:\mydocuments\letters\*", "c:\tempfolder\"
```

Но вы не можете использовать такой шаблон:

```
FileSystemObject.CopyFolder "c:\mydocuments\*\*", "c:\tempfolder\"
```

Если **Источник** содержит подстановочные символы или **Приёмник** заканчивается символом разделения пути (\), это подразумевает, что **Приёмник** – это существующая папка, в которую будут копироваться каталоги, соответствующие шаблону. Иначе предполагается, что **Приёмник** – это имя каталога, который будет создан и в который будет скопирован каталог-источник. В любом случае могут произойти следующие события при копировании каталога:

- Если **Приёмник** не существует, то каталог и всё его содержимое копируется в **Источник**. То есть каталог будет скопирован в ту же папку, на которую указывает **Источник**.
- Если **Приёмник** – это существующий файл, то происходит ошибка.
- Если **Приёмник** – это директория, то выполняется копирование папки и всего её содержимого. Если файл из **Источник** уже имеется в **Приёмнике**, происходит ошибка, если флаг **Перезапись** равен False. Если флаг **Перезапись** равен True, то существующий файл перезаписывается.
- Если **Приёмник** – это существующая директория с атрибутом «только чтение» или в приёмнике имеется файл с таким атрибутом, который необходимо заменить, то происходит ошибка, если флаг **Перезапись** равен False.

Ошибка также случается, если **Источник** использует подстановочные символы, которым не соответствуют никакие каталоги. Метод **CopyFolder** прекращает работу при обнаружении первой ошибки. Все выполненные перед ошибкой операции не отменяются.

Метод CreateTextFile

Метод создаёт файл с указанным именем и возвращает объект **TextStream**, который может быть использован для чтения и записи файла. Синтаксис:

```
Объект.CreateTextFile(ИмяФайла[, Перезапись[, Юникод]])
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
ИмяФайла	Обязательный элемент. Строковое выражение, которое определяет имя создаваемого файла.
Перезапись	Не обязательный элемент. Логическое значение, которое определяет поведение метода, если файл, указанный в параметре **ИмяФайла**, уже существует. Если равно True, то файл будет перезаписан, если равно False, то файл не будет перезаписан. По умолчанию (если параметр не указан) существующий файл не перезаписывается.
Юникод	Не обязательный параметр. Логическое значение, которое определяет, в какой кодировке будет создан новый файл – Unicode или ASCII. Если равно True, то создаётся файл в кодировке Unicode, иначе – в кодировке ASCII. Если не указано, то создаётся ASCII-файл.

ПРИМЕЧАНИЕ

В следующем примере показано, как использовать метод CreateTextFile для создания и открытия текстового файла:

```
Sub CreateAfile
  Dim fso, MyFile
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set MyFile = fso.CreateTextFile("c:\testfile.txt", True)
  MyFile.WriteLine("Это проверка.")
  MyFile.Close
End Sub
```

Если параметр **Перезапись** равен False или не указан, то в том случае, если такой файл уже существует, произойдёт ошибка.

Метод DeleteFile

Метод удаляет указанный файл. Синтаксис:

```
Объект.DeleteFile ИмяФайла[, УдалятьВсё]
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
ИмяФайла	Обязательный элемент. Строковое выражение, которое определяет имя удаляемого файла.
УдалятьВсё	Не обязательный элемент. Логическое значение, которое определяет поведение метода, если файл, указанный в параметре **ИмяФайла**, имеет атрибут ТОЛЬКО ЧТЕНИЕ. Если равно True, то файл будет удалён, если равно False (по умолчанию), то файл не будет удалён.

ПРИМЕЧАНИЕ

Если указанный файл не найден, то происходит ошибка. Метод **DeleteFile** прекращает работу при обнаружении первой ошибки. При этом уже удалённые файлы не восстанавливаются. В следующем примере показано применение метода **DeleteFile**:

```
Sub DeleteAFile(filespec)
  Dim fso
  Set fso = CreateObject("Scripting.FileSystemObject")
  fso.DeleteFile(filespec)
End Sub
```

Метод DeleteFolder

Метод удаляет указанный каталог. Синтаксис:

```
Объект.DeleteFolder ИмяКаталога[, УдалятьВсё]
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
ИмяКаталога	Обязательный элемент. Строковое выражение, которое определяет имя удаляемого каталога.
УдалятьВсё	Не обязательный элемент. Логическое значение, которое определяет поведение метода, если каталог, указанный в параметре **ИмяКаталога**, имеет атрибут ТОЛЬКО ЧТЕНИЕ. Если равно True, то каталог будет удалён, если равно False (по умолчанию), то каталог не будет удалён.

ПРИМЕЧАНИЕ

Метод **DeleteFolder** не делает различия между каталогами, в которых имеется содержимое и пустыми каталогами. Указанный каталог удаляется, независимо от того, является ли он пустым или имеет содержимое.

Если указанный каталог не найден, то происходит ошибка. Метод **DeleteFolder** прекращает работу при обнаружении первой ошибки. При этом уже удалённые каталоги не восстанавливаются.

В следующем примере показано применение метода **DeleteFolder**:

```
Sub DeleteAFolder(filespec)
  Dim fso
  Set fso = CreateObject("Scripting.FileSystemObject")
  fso.DeleteFolder(filespec)
End Sub
```

Метод DriveExists

Метод возвращает TRUE, если диск существует. Если диск не существует, то метод возвращает FALSE. Синтаксис:

```
Объект.DriveExists(ИмяДиска)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
ИмяДиска	Обязательный элемент. Имя диска или полный путь к диску (только буква, например, “C”, или полный путь, например, “C:\”).

ПРИМЕЧАНИЕ

Для дисководов и других съёмных дисков метод **DriveExists** возвращает TRUE, даже если в дисководе нет диска. Используёте свойство **IsReady** объекта **Drive** для определения готовности дисковода.

В следующем примере показано применение метода **DriveExists**:

```
Function ReportDriveStatus(drv)
  Dim fso, msg
  Set fso = CreateObject("Scripting.FileSystemObject")
  If fso.DriveExists(drv) Then
    msg = ("Диск " & UCase(drv) & " существует.")
  Else
    msg = ("Диск " & UCase(drv) & " НЕ существует.")
  End If
  ReportDriveStatus = msg
End Function
```

Метод FileExists

Метод возвращает TRUE, если указанный файл существует. Если файл не существует, то метод возвращает FALSE. Синтаксис:

```
Объект.FileExists(ИмяФайла)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
ИмяФайла	Обязательный элемент. Имя файла, существование которого требуется проверить. Требуется указывать полный путь (абсолютный или относительный), кроме случаев, когда нужно проверить только текущий каталог.

ПРИМЕЧАНИЕ

В следующем примере показано применение метода **FileExists**:

```
Function ReportFileStatus(filespec)
  Dim fso, msg
  Set fso = CreateObject("Scripting.FileSystemObject")
  If (fso.FileExists(filespec)) Then
    msg = filespec & " существует."
  Else
    msg = filespec & " НЕ существует."
  End If
  ReportFileStatus = msg
End Function
```

Метод FolderExists

Метод возвращает TRUE, если указанный каталог существует. Если каталог не существует, то метод возвращает FALSE. Синтаксис:

```
Объект.FolderExists(ИмяПапки)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
ИмяПапки	Обязательный элемент. Имя каталога, существование которого требуется проверить. Требуется указывать полный путь (абсолютный или относительный), кроме случаев, когда нужно проверить только текущий каталог.

ПРИМЕЧАНИЕ

В следующем примере показано применение метода **FolderExists**:

```
Function ReportFolderStatus(fldr)
  Dim fso, msg
  Set fso = CreateObject("Scripting.FileSystemObject")
  If (fso.FolderExists(fldr)) Then
    msg = fldr & " существует."
  Else
    msg = fldr & " НЕ существует."
  End If
  ReportFolderStatus = msg
End Function
```

Метод GetAbsolutePathName

Метод определяет и возвращает полный и точный путь по части пути. Синтаксис:

```
Объект.GetAbsolutePathName(Путь)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
Путь	Обязательный элемент. Часть пути, по которой выполняется определение полного пути.

ПРИМЕЧАНИЕ

Путь является полным и точным, если он записан как полная ссылка, начиная от корневого каталога указанного диска.

Например, если сценарий запускается из каталога **c:\mydocuments\reports**, то метод будет возвращать значения, приведённые ниже в таблице:

Путь	Возвращаемое значение
"c:"	"c:\mydocuments\reports"
"c:.."	"c:\mydocuments"
"c:\\\"	"c:\"
"c:*.*\may97"	"c:\mydocuments\reports*.*\may97"
"region1"	"c:\mydocuments\reports\region1"
"c:\..\..\mydocuments"	"c:\mydocuments"

В следующем примере показано применение метода **GetAbsolutePathName**:

```
Dim fso
Set fso = CreateObject("Scripting.FileSystemObject")
MsgBox fso.GetAbsolutePathName("HELP-TEST.HLP")
```

В этом примере мы определяем полный путь к файлу HELP-TEST.HLP, который должен находиться в том же каталоге, из которого запускается сценарий. Причём этот файл может не существовать. В этом случае сценарий вернёт следующее (если он запускается из каталога **c:\mydocuments\reports**):

```
c:\mydocuments\reports\HELP-TEST.HLP
```

Если же необходимо определить только путь к текущему каталогу, то в качестве параметра можно передать пустую строку:

```
MsgBox fso.GetAbsolutePathName("")
```

В этом случае для нашего примера сценарий вернёт следующее значение:

```
c:\mydocuments\reports
```

Метод GetBaseName

Метод возвращает строку, содержащую имя указанного файла без расширения или имя папки из указанного пути. Синтаксис:

```
Объект.GetBaseName(Путь)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
Путь	Обязательный элемент. Путь к файлу или часть пути.

ПРИМЕЧАНИЯ

Метод **GetBaseName** возвращает пустую строку если параметр **Путь** не содержит правильного пути к файлу или папке.

Учтите, что метод **GetBaseName** работает только с указанной в параметре строкой. Он не пытается проверить существование указанного файла или папки и правильность пути.

В следующем примере показано применение метода **GetBaseName**:

```
Function GetTheBase(filespec)
  Dim fso
  Set fso = CreateObject("Scripting.FileSystemObject")
  GetTheBase = fso.GetBaseName(filespec)
End Function
```

Метод GetDrive

Метод возвращает ссылку на объект **Drive**. В качестве параметра принимается путь к диску. Синтаксис:

```
Объект.GetDrive Диск
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
Диск	Обязательный элемент. Этот параметр может быть буквой диска (c), буквой диска с двоеточием (c:), буквой диска с двоеточием и разделителем (c:\) или любым сетевым именем, которое указывает на общедоступный ресурс (\\computer2\share1).

ПРИМЕЧАНИЯ

В случае с сетевыми ресурсами, перед вызовом метода убедитесь, что ресурс существует.

Ошибка происходит, если диск, указанный в параметре **Диск**, по каким-либо причинам не доступен или не существует. При вызове метода GetDrive в качестве параметра можно передавать полный путь к папке или файлу. В этом случае рекомендуется использовать для получения правильного имени диска следующий синтаксис:

```
DriveSpec = GetDriveName(GetAbsolutePathName(Path))
```

В следующем примере показано применение метода **GetDrive**:

```
Function ShowFreeSpace(drvPath)
  Dim fso, d, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set d = fso.GetDrive(fso.GetDriveName(drvPath))
  s = "Диск " & UCase(drvPath) & " - "
  s = s & d.VolumeName  & vbCrLf
  s = s & "Свободно: " & FormatNumber(d.FreeSpace/1024, 0)
  s = s & " КБ"
  ShowFreeSpace = s
End Function
```

Метод GetDriveName

Метод возвращает строку, содержащую имя диска из указанного пути. Синтаксис:

```
Объект.GetDriveName(Путь)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
Путь	Обязательный элемент. Путь к файлу, в котором содержится имя диска.

ПРИМЕЧАНИЯ

Метод **GetDriveName** возвращает пустую строку, если диск не может быть определён.

В следующем примере показано применение метода **GetDriveName**:

```
Function GetAName(DriveSpec)
  Dim fso
  Set fso = CreateObject("Scripting.FileSystemObject")
  GetAName = fso.GetDriveName(Drivespec)
End Function
```

Метод GetExtensionName

Метод возвращает строку, содержащую расширение имени файла из указанного пути. Синтаксис:

```
Объект.GetExtensionName(Путь)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
Путь	Обязательный элемент. Путь к файлу, расширение которого требуется получить.

ПРИМЕЧАНИЯ

Для сетевых дисков корневой каталог является частью пути.

Метод **GetExtensionName** возвращает пустую строку, если в пути нет расширения файла.

В следующем примере показано применение метода **GetExtensionName**:

```
Function GetAnExtension(DriveSpec)
  Dim fso
  Set fso = CreateObject("Scripting.FileSystemObject")
  GetAnExtension = fso.GetExtensionName(Drivespec)
End Function
```

Метод GetFile

Метод возвращает объект **File**, связанный с файлом, указанным в параметре. Синтаксис:

```
Объект.GetFile(Путь)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
Путь	Обязательный элемент. Путь к файлу (абсолютный или относительный), ссылку на который требуется получить.

ПРИМЕЧАНИЯ

Если указанный файл не существует, то происходит ошибка.

В следующем примере показано применение метода **GetFile**:

```
Function ShowFileAccessInfo(filespec)
  Dim fso, f, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFile(filespec)
  s = f.Path & vbCrLf
  s = s & "Создан : " & f.DateCreated & vbCrLf
  s = s & "Последний доступ : " & f.DateLastAccessed & vbCrLf
  s = s & "Последнее изменение : " & f.DateLastModified
  ShowFileAccessInfo = s
End Function
```

Метод GetFileName

Метод возвращает последнее имя файла или папки в указанном пути. Синтаксис:

```
Объект.GetFileName(Путь)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
Путь	Обязательный элемент. Путь к файлу (абсолютный или относительный).

ПРИМЕЧАНИЯ

Метод **GetFileName** возвращает пустую строку, если указанный путь не заканчивается именем папки или файла.

В следующем примере показано применение метода **GetFileName**:

```
Function GetAName(DriveSpec)
  Dim fso
  Set fso = CreateObject("Scripting.FileSystemObject")
  GetAName = fso.GetFileName(DriveSpec)
End Function
```

Учтите, что метод **GetFileName** работает только со строкой, переданной через параметр **Путь**. Метод не проверяет правильность пути и не проверяет существование файла.

Метод GetFolder

Метод возвращает объект **Folder**, ссылающийся на указанную папку. Синтаксис:

```
Объект.GetFolder(Путь)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
Путь	Обязательный элемент. Путь к папке (абсолютный или относительный).

ПРИМЕЧАНИЯ

Если указанная папка не существует, то происходит ошибка.

В следующем примере показано применение метода **GetFolder**:

```
Sub AddNewFolder(path, folderName)
  Dim fso, f, fc, nf
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFolder(path)
  Set fc = f.SubFolders
  If folderName <> "" Then           'Если имя задано, то
    Set nf = fc.Add(folderName)      'создать папку с этим именем
  Else                               'Если имя НЕ задано, то
    Set nf = fc.Add("Новая папка")   'создать новую папку
  End If
End Sub
```

Метод GetParentFolderName

Метод возвращает строку, содержащую имя родительского каталога для файла или папки, указанных в пути. Синтаксис:

```
Объект.GetParentFolderName(Путь)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
Путь	Обязательный элемент. Путь к папке или файлу, для которого требуется получить родительский каталог.

ПРИМЕЧАНИЯ

Метод **GetParentFolderName** возвращает пустую строку, если для указанного файла (каталога) не существует родительского каталога.

В следующем примере показано применение метода **GetParentFolderName**:

```
Function GetTheParent(DriveSpec)
  Dim fso
  Set fso = CreateObject("Scripting.FileSystemObject")
  GetTheParent = fso.GetParentFolderName(Drivespec)
End Function
```

Учтите, что метод **GetParentFolderName** работает только со строкой, переданной через параметр **Путь**. Метод не проверяет правильность пути и не проверяет существование файла (каталога).

Метод GetSpecialFolder

Метод возвращает системную папку. Синтаксис:

```
Объект.GetSpecialFolder(Папка)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
Папка	Обязательный элемент. Тип системной папки. Может принимать значения, описанные в таблице **Параметры**.

Параметры

Параметр **Папка** может принимать следующие значения:

Константа	Значение	Описание
WindowsFolder	0	Папка Windows содержит файлы, установленные операционной системой Windows
SystemFolder	1	Папка System содержит библиотеки, шрифты и драйверы
TemporaryFolder	2	Папка Temp используется для хранения временных файлов. Путь к этой папке находится в среде окружения в переменной TMP.

В следующем примере показано применение метода **GetSpecialFolder**:

```
Function GetSysFolder
   Dim fso, sfolder, p
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set sfolder = fso.GetSpecialFolder(0)
   p = sfolder.Path
   GetSysFolder = p
End Function
```

Метод GetTempName

Метод возвращает сгенерированное случайным образом имя временных папки или файла для выполнения операции, которые требуют создания временных папок или файлов. Синтаксис:

```
Объект.GetTempName
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.

ПРИМЕЧАНИЕ

Метод **GetTempName** не создаёт файл, а только предоставляет имя временного файла, которое затем можно использовать при создании файла методом **CreateTextFile**.

В следующем примере показано применение метода **GetTempName**:

```
Dim fso, tempfile
Set fso = CreateObject("Scripting.FileSystemObject")

Function CreateTempFile
   Dim tfolder, tname, tfile
   Const TemporaryFolder = 2
   Set tfolder = fso.GetSpecialFolder(TemporaryFolder)
   tname = fso.GetTempName
   Set tfile = tfolder.CreateTextFile(tname)
   Set CreateTempFile = tfile
End Function

Set tempfile = CreateTempFile
tempfile.WriteLine "Hello World"
tempfile.Close
```

Метод MoveFile

Метод перемещает один или несколько файлов из одного местоположения в другое.
Синтаксис:

```
Объект.MoveFile Источник, Приёмник
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
Источник	Обязательный элемент. Путь к файлу или к файлам, которые требуется переместить. В качестве этого параметра может быть строка, в которой допускается применение подстановочных символов только в последней части пути.
Приёмник	Обязательный элемент. Путь, куда файл или файлы будут перемещены. Этот параметр не может содержать подстановочных символов.

ПРИМЕЧАНИЯ

Если **Источник** содержит подстановочные символы или **Приёмник** заканчивается символом разделителя (\), это подразумевает, что **Приёмник** указывает на существующую папку в которую должны быть скопированы несколько файлов. В противном случае **Приёмник** расценивается как имя нового файла, который требуется создать. Во всех случаях возможны следующие варианты при перемещении файлов:

- Если **Приёмник** не существует, то файлы будут перемещены. Это общий случай.
- Если **Приёмник** – это существующий файл, то произойдёт ошибка.
- Если **Приёмник** – это каталог, то произойдёт ошибка.

Ошибка также происходит, если подстановочные символы, которые используются в **Источнике**, не соответствуют ни одному файлу. Метод **MoveFile** прекращает выполнение при первой ошибке, при этом изменения, которые были сделаны методом до появления ошибки, не отменяются.

В следующем примере показано применение метода **MoveFile**:

```
Sub MoveAFile(Drivespec)
  Dim fso
  Set fso = CreateObject("Scripting.FileSystemObject")
  fso.MoveFile Drivespec, "c:\windows\desktop\"
End Sub
```

ВАЖНО

Этот метод перемещает файлы из одного тома в другой, только если это поддерживается операционной системой.

Метод MoveFolder

Метод перемещает одну или несколько папок из одного местоположения в другое.
Синтаксис:

```
Объект.MoveFolder Источник, Приёмник
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
Источник	Обязательный элемент. Путь к папке или к папкам, которые требуется переместить. В качестве этого параметра может быть строка, в которой допускается применение подстановочных символов только в последней части пути.
Приёмник	Обязательный элемент. Путь, куда папка или папки будут перемещены. Этот параметр не может содержать подстановочных символов.

ПРИМЕЧАНИЯ

Если **Источник** содержит подстановочные символы или **Приёмник** заканчивается символом разделителя (\), это подразумевает, что **Приёмник** указывает на существующую папку в которую должны быть скопированы несколько папок. В противном случае **Приёмник** расценивается как имя новой папки, которую требуется создать. Во всех случаях возможны следующие варианты при перемещении папок:

- Если **Приёмник** не существует, то папки будут перемещены. Это общий случай.
- Если **Приёмник** – это существующий файл, то произойдёт ошибка.
- Если **Приёмник** – это каталог, то произойдёт ошибка.

Ошибка также происходит, если подстановочные символы, которые используются в **Источнике**, не соответствуют ни одной папке. Метод **MoveFolder** прекращает выполнение при первой ошибке, при этом изменения, которые были сделаны методом до появления ошибки, не отменяются.

В следующем примере показано применение метода **MoveFolder**:

```
Sub MoveAFolder(Drivespec)
  Dim fso
  Set fso = CreateObject("Scripting.FileSystemObject")
  fso.MoveFolder Drivespec, "c:\windows\desktop\"
End Sub
```

ВАЖНО

Этот метод перемещает папки из одного тома в другой, только если это поддерживается операционной системой.

Метод OpenTextFile

Метод открывает указанный файл и возвращает объект **TextStream**, который может использоваться для записи, чтения или добавления информации в файл. Синтаксис:

```
Объект.OpenTextFile(ИмяФайла[, Режим[, Создание[, Формат]]])
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **FileSystemObject**.
ИмяФайла	Обязательный элемент. Строковое выражение, которое идентифицирует открываемый файл.
Режим	Необязательный параметр. Указывает режим ввода-вывода. Может быть одной из трёх констант: **ForReading**, **ForWriting** или **ForAppending**.
Создание	Необязательный параметр. Логическое значение, которое определяет, нужно ли создавать новый файл, если файл, указанный в параметре **ИмяФайла** не существует. Если значение равно TRUE, то новый файл создаётся, если FALSE (по умолчанию), то не создаётся.
Формат	Необязательный параметр. Одно из трёх значений, используемое для определения формата открываемого файла. Если не указано, то файл открывается как ASCII.

Параметры

Параметр **Режим** может принимать следующие значения:

Константа	Значение	Описание
ForReading	1	Открыть файл только для чтения. Вы не можете записывать данные в этот файл.
ForWriting	2	Открыть файл только для записи. Вы не можете читать данные из этого файла.
ForAppending	8	Открыть файл и записать данные в конец файла.

Параметр **Формат** может принимать следующие значения:

Константа	Значение	Описание
TristateUseDefault	-2	Открыть файл, используя системные настройки по умолчанию.
TristateTrue	-1	Открыть файл как Unicode.
TristateFalse	0	Открыть файл как ASCII.

В следующем примере показано применение метода **OpenTextFile**:

```
Sub OpenTextFileTest
  Const ForReading = 1, ForWriting = 2, ForAppending = 8
  Dim fso, f
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.OpenTextFile("testfile.txt", ForWriting, True)
  f.Write "Привет, Мир!"
  f.Close
End Sub
```

5.7.5. Объект Drive

Объект предоставляет доступ к разделу диска или к сетевому общедоступному диску. В следующем примере показано использование объекта **Drive** для получения свойств диска:

```
Function ShowFreeSpace(drvPath)
  Dim fso, d, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set d = fso.GetDrive(fso.GetDriveName(drvPath))
  s = "Диск " & UCase(drvPath) & " - "
  s = s & d.VolumeName  & vbCrLf
  s = s & "Свободное место: " & FormatNumber(d.FreeSpace/1024, 0)
  s = s & " КБ"
  ShowFreeSpace = s
End Function

MsgBox ShowFreeSpace("C:\")
```

5.7.5.1. Свойства объекта Drive

Свойство AvailableSpace

Возвращает количество свободного доступного пользователю пространства на диске или сетевом общедоступном ресурсе. Синтаксис:

```
Объект.AvailableSpace
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Drive**.

ПРИМЕЧАНИЕ

Значение, возвращаемое свойством **AvailableSpace** обычно равно значению, которое можно получить через свойство **FreeSpace**. Различия могут возникнуть на разных компьютерных системах, которые поддерживают квоты.

В следующем примере показано использование свойства **AvailableSpace**:

```
Function ShowAvailableSpace(drvPath)
  Dim fso, d, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set d = fso.GetDrive(fso.GetDriveName(drvPath))
  s = "Диск " & UCase(drvPath) & " - "
  s = s & d.VolumeName  & vbCrLf
  s = s & "Доступное место: " & FormatNumber(d.AvailableSpace/1024, 0)
  s = s & " КБ"
  ShowAvailableSpace = s
End Function
```

Свойство DriveLetter

Возвращает букву физического локального диска или сетевого ресурса. Только для чтения. Синтаксис:

```
Объект.DriveLetter
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Drive**.

ПРИМЕЧАНИЕ

Свойство **DriveLetter** возвращает пустую строку (“”), если указанный диск не связан с буквой диска, например, если это сетевой ресурс, которому не была назначена буква. В следующем примере показано использование свойства **DriveLetter**:

```
Function ShowDriveLetter(drvPath)
  Dim fso, d, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set d = fso.GetDrive(fso.GetDriveName(drvPath))
  s = "Диск " & d.DriveLetter & ": - "
  s = s & d.VolumeName & vbCrLf
  s = s & "Свободное место: " & FormatNumber(d.FreeSpace/1024, 0)
  s = s & " КБ"
  ShowDriveLetter = s
End Function
```

Свойство DriveType

Возвращает значение, определяющее тип указанного диска. Синтаксис:

```
Объект.DriveType
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Drive**.

В следующем примере показано использование свойства **DriveType**:

```
Function ShowDriveType(drvpath)
  Dim fso, d, t
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set d = fso.GetDrive(drvpath)
  Select Case d.DriveType
    Case 0: t = "Неизвестный"
    Case 1: t = "Съёмный"
    Case 2: t = "Несъёмный"
    Case 3: t = "Сетевой"
    Case 4: t = "CD-ROM"
    Case 5: t = "RAM-диск"
  End Select
  ShowDriveType = "Диск " & d.DriveLetter & ": - " & t
End Function
```

Свойство FileSystem

Возвращает тип файловой системы, которая используется на указанном диске. Синтаксис:

```
Объект.FileSystem
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Drive**.

Типы файловых систем, которые может вернуть это свойство: FAT, NTFS или CDFS. В следующем примере показано использование свойства **FileSystem**:

```
Function ShowFileSystemType(drvspec)
  Dim fso,d
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set d = fso.GetDrive(drvspec)
  ShowFileSystemType = "Файловая система: " & d.FileSystem
End Function
```

Свойство FreeSpace

Возвращает количество свободного доступного пользователю пространства на диске или сетевом общедоступном ресурсе. Синтаксис:

```
Объект.FreeSpace
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Drive**.

Значение, возвращаемое свойством **FreeSpace** обычно равно значению, которое можно получить через свойство **AvailableSpace**. Различия могут возникнуть на разных компьютерных системах, которые поддерживают квоты.

В следующем примере показано использование свойства **FreeSpace**:

```
Function ShowFreeSpace(drvPath)
  Dim fso, d, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set d = fso.GetDrive(fso.GetDriveName(drvPath))
  s = "Диск " & UCase(drvPath) & " - "
  s = s & d.VolumeName  & vbCrLf
  s = s & "Свободное пространство: " & FormatNumber(d.FreeSpace/1024, 0)
  s = s & " КБ"
  ShowFreeSpace = s
End Function
```

Свойство IsReady

Возвращает TRUE, если указанный диск готов к использованию, иначе возвращает FALSE. Синтаксис:

```
Объект.IsReady
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Drive**.

Для съёмных дисков и CD-ROM-дисководов свойство **IsReady** возвращает TRUE только когда диск вставлен в дисковод и готов к работе. В следующем примере показано использование свойства **IsReady**:

```
Function ShowDriveInfo(drvpath)
  Dim fso, d, s, t
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set d = fso.GetDrive(drvpath)
  Select Case d.DriveType
    Case 0: t = "Неизвестный"
    Case 1: t = "Съёмный"
    Case 2: t = "Несъёмный"
    Case 3: t = "Сетевой"
    Case 4: t = "CD-ROM"
    Case 5: t = "RAM-диск"
  End Select
  s = "Диск " & d.DriveLetter & ": - " & t
  If d.IsReady Then
    s = s & vbCrLf & "Диск готов."
  Else
    s = s & vbCrLf & "Диск НЕ готов."
  End If
  ShowDriveInfo = s
End Function
```

Свойство Path

Возвращает путь к указанному файлу, папке или диску. Синтаксис:

```
Объект.Path
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **File**, **Folder** или **Drive**.

Для буквы диска корень диска не включается в путь. Например, путь к диску C – это C:, а не C:\. В следующем примере показано использование свойства **Path** с объектом **File**:

```
Function ShowFileAccessInfo(filespec)
  Dim fso, d, f, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFile(filespec)
  s = UCase(f.Path) & vbCrLf
  s = s & "Создан: " & f.DateCreated & vbCrLf
  s = s & "Последний доступ: " & f.DateLastAccessed & vbCrLf
  s = s & "Последние изменения: " & f.DateLastModified
  ShowFileAccessInfo = s
End Function
```

Свойство RootFolder

Возвращает объект **Folder**, связанный с корневым каталогом указанного диска. Синтаксис:

```
Объект.RootFolder
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Drive**.

Ко всем файлам и папкам, имеющимся на указанном диске, можно получить доступ, используя объект **Folder**, полученный через это свойство. В следующем примере показано использование свойства **RootFolder**:

```
Function ShowRootFolder(drvspec)
  Dim fso, f
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetDrive(drvspec)
  ShowRootFolder = f.RootFolder
End Function
```

Свойство SerialNumber

Возвращает десятичный серийный номер, используемый для уникальной идентификации диска. Синтаксис:

```
Объект.SerialNumber
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Drive**.

Вы можете использовать свойство **SerialNumber** для того, чтобы убедиться, что в дисковод вставлен нужный диск. В следующем примере показано использование свойства **SerialNumber**:

```
Function ShowSerialNum(drvpath)
  Dim fso, d, s, t
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set d = fso.GetDrive(fso.GetDriveName(fso.GetAbsolutePathName(drvpath)))
  Select Case d.DriveType
    Case 0: t = "Неизвестный"
    Case 1: t = "Съёмный"
    Case 2: t = "Несъёмный"
    Case 3: t = "Сетевой"
    Case 4: t = "CD-ROM"
    Case 5: t = "RAM-диск"
  End Select
  s = "Диск " & d.DriveLetter & ": - " & t
  s = s & vbCrLf & "Серийный номер: " & d.SerialNumber
  ShowSerialNum = s
End Function
```

Свойство ShareName

Возвращает сетевое имя указанного диска. Синтаксис:

```
Объект.ShareName
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Drive**.

Если объект НЕ является сетевым диском, то свойство **ShareName** вернёт пустую строку (“”). В следующем примере показано использование свойства **ShareName**:

```
Function ShowShareName(drvpath)
  Dim fso, d
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set d = fso.GetDrive(fso.GetDriveName(fso.GetAbsolutePathName(drvpath)))
  ShowShareName = "Диск " & d.DriveLetter & ": - " & d.ShareName
End Function
```

Свойство TotalSize

Возвращает общий объём указанного диска или сетевого ресурса. Синтаксис:

```
Объект.TotalSize
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Drive**.

В следующем примере показано использование свойства **TotalSize**:

```
Function ShowSpaceInfo(drvpath)
  Dim fso, d, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set d = fso.GetDrive(fso.GetDriveName(fso.GetAbsolutePathName(drvpath)))
  s = "Диск " & d.DriveLetter & ":"
  s = s & vbCrLf
  s = s & "Общий объём: " & FormatNumber(d.TotalSize/1024, 0) & " КБ"
  s = s & vbCrLf
  s = s & "Доступно: " & FormatNumber(d.AvailableSpace/1024, 0) & " КБ"
  ShowSpaceInfo = s
End Function
```

Свойство VolumeName

Возвращает общий объём указанного диска или сетевого ресурса. Синтаксис:

```
Объект.VolumeName [= НовоеИмя]
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Drive**.
НовоеИмя	Не обязательный параметр. Если имеется, то это будет новое имя указанного объекта.

В следующем примере показано использование свойства **VolumeName**:

```
Function ShowVolumeInfo(drvpath)
  Dim fso, d, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set d = fso.GetDrive(fso.GetDriveName(fso.GetAbsolutePathName(drvpath)))
  s = "Диск " & d.DriveLetter & ": - " & d.VolumeName
  ShowVolumeInfo = s
End Function
```

5.7.6. Объект File

Объект предоставляет ко всем свойствам файла. Следующий пример показывает применение объекта **File** и получение свойств файла:

```
Function ShowFreeSpace(drvPath)
  Dim fso, d, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set d = fso.GetDrive(fso.GetDriveName(drvPath))
  s = "Диск " & UCase(drvPath) & " - "
  s = s & d.VolumeName  & vbCrLf
  s = s & "Свободное место: " & FormatNumber(d.FreeSpace/1024, 0)
  s = s & " КБ"
  ShowFreeSpace = s
End Function

MsgBox ShowFreeSpace("C:\")
```

5.7.6.1. Свойства объекта File

Свойство Attributes

Устанавливает или возвращает атрибуты файла или папки. Возможны два режима: только чтение или чтение/запись. Режим определяется атрибутом. Синтаксис:

```
Объект.Attributes [= НовыйАтрибут]
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **File** или **Folder**.
НовыйАтрибут	Не обязательный параметр. Если указан, то НовыйАтрибут – это новое значение атрибутов для указанного объекта (файла или папки).

Значения атрибутов

Параметр НовыйАтрибут может иметь любое из значений, перечисленных ниже в таблице или любую комбинацию из этих значений.

Константа	Значение	Описание
Normal	0	Обычный файл. Никакие атрибуты не установлены.
ReadOnly	1	Файл только для чтения. Атрибут для чтения/записи.
Hidden	2	Скрытый файл. Атрибут для чтения/записи.
System	4	Системный файл. Атрибут для чтения/записи.
Volume	8	Метка тома диска. Атрибут только для чтения.
Directory	16	Папка или каталог. Атрибут только для чтения.
Archive	32	Архивный файл. Атрибут для чтения/записи.
Alias	64	Ссылка или ярлык. Атрибут только для чтения.
Compressed	128	Сжатый файл. Атрибут только для чтения.

В следующем примере показано использование свойства **Attributes**:

```
Function ToggleArchiveBit(filespec)
  Dim fso, f
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFile(filespec)
  If f.attributes and 32 Then
    f.attributes = f.attributes - 32
    ToggleArchiveBit = "Бит архивного файла очищен."
  Else
    f.attributes = f.attributes + 32
    ToggleArchiveBit = "Бит архивного файла установлен."
  End If
End Function
```

Свойство DateCreated

Возвращает дату и время, когда был создан указанный файл или папка. Синтаксис:

```
Объект.DateCreated
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **File** или **Folder**.

В следующем примере показано использование свойства **DateCreated**:

```
Function ShowFileInfo(filespec)
  Dim fso, f
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFile(filespec)
  ShowFileInfo = "Создан: " & f.DateCreated
End Function
```

Свойство DateLastAccessed

Возвращает дату и время, когда последний раз было выполнено обращение к указанному файлу или папке. Синтаксис:

```
Объект.DateLastAccessed
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **File** или **Folder**.

В следующем примере показано использование свойства **DateLastAccessed**:

```
Function ShowFileAccessInfo(filespec)
  Dim fso, f, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFile(filespec)
  s = UCase(filespec) & vbCrLf
  s = s & "Создан: " & f.DateCreated & vbCrLf
  s = s & "Последний доступ: " & f.DateLastAccessed & vbCrLf
  s = s & "Последние изменения: " & f.DateLastModified
  ShowFileAccessInfo = s
End Function
```

ВАЖНО!
Значение этого свойства зависит от поведения операционной системы. Если операционная система не поддерживает функцию предоставления времени, то свойство ничего не вернёт.

Свойство DateLastModified

Возвращает дату и время, когда последний раз указанный файл или папка были изменены.
Синтаксис:

```
Объект.DateLastModified
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **File** или **Folder**.

В следующем примере показано использование свойства **DateLastModified**:

```
Function ShowFileAccessInfo(filespec)
  Dim fso, f, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFile(filespec)
  s = UCase(filespec) & vbCrLf
  s = s & "Создан: " & f.DateCreated & vbCrLf
  s = s & "Последний доступ: " & f.DateLastAccessed & vbCrLf
  s = s & "Последние изменения: " & f.DateLastModified
  ShowFileAccessInfo = s
End Function
```

Свойство Drive

Возвращает букву диска, на котором находится указанный файл или папка. Только чтение.
Синтаксис:

```
Объект.Drive
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **File** или **Folder**.

В следующем примере показано использование свойства **Drive**:

```
Function ShowFileDriveInfo(filespec)
  Dim fso, f, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFile(filespec)
  s = f.Name & " на диске " & UCase(f.Drive) & vbCrLf
  ShowFileDriveInfo = s
End Function
```

Свойство Name

Устанавливает или возвращает имя указанного файла или папки. Чтение/запись. Синтаксис:

```
Объект.Name [= НовоеИмя]
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **File** или **Folder**.
НовоеИмя	Не обязательный параметр. Если указан, то это будет новое имя указанного **Объекта**.

В следующем примере показано использование свойства **Name**:

```
Function ShowFileDriveInfo(filespec)
  Dim fso, f, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFile(filespec)
  s = f.Name & " на диске " & UCase(f.Drive) & vbCrLf
  ShowFileDriveInfo = s
End Function
```

Свойство ParentFolder

Возвращает объект **Folder** – родительский каталог указанного файла или папки. Только чтение. Синтаксис:

```
Объект.ParentFolder
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **File** или **Folder**.

В следующем примере показано использование свойства **ParentFolder**:

```
Function ShowParentFolder(filespec)
  Dim fso, f, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFile(filespec)
  s = "Файл " & UCase(f.Name) & " находится в папке " &
UCase(f.ParentFolder)
  ShowParentFolder = s
End Function
```

Свойство Path

Возвращает полный путь для указанного файла, папки или диска. Синтаксис:

```
Объект.Path
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **File**, **Folder** или **Drive**.

В случае для объекта **Drive** символ коневого каталога не содержится в результате. Например, путь для диска C будет C:, а не C:\. В следующем примере показано использование свойства **Path**:

```
Function ShowPath(filespec)
  Dim fso, f, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFile(filespec)
  s = "Полный путь к файлу " & UCase(f.Path)
  ShowPath = s
End Function
```

Свойство ShortName

Возвращает сокращённое (обрезанное) имя указанного файла или папки. Используется программами, которые требуют выполнения соглашения об именовании файлов в формате 8.3 (8 символов имя, 3 символа - расширение). Синтаксис:

```
Объект.ShortName
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **File** или **Folder**.

В следующем примере показано использование свойства **ShortName**:

```
Function ShowShortName(filespec)
  Dim fso, f, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFile(filespec)
  s = "Сокращённое имя для файла "  & UCase(f.Name) & vbCrLf
  s = s & " - это " & f.ShortName
  ShowShortName = s
End Function
```

Свойство ShortPath

Возвращает сокращённый путь для указанного файла или папки. То есть путь, где все имена каталогов и имя файла соответствуют соглашению об именовании файлов в формате 8.3 (8 символов имя, 3 символа - расширение). Синтаксис:

```
Объект.ShortPath
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **File** или **Folder**.

В следующем примере показано использование свойства **ShortPath**:

```
Function ShowShortPath(filespec)
  Dim fso, f, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFile(filespec)
  s = "Сокращённый путь для файла " & UCase(f.Name) & vbCrLf
  s = s & " - это " & f.ShortPath
  ShowShortPath = s
End Function
```

Свойство Size

Для файлов возвращает размер указанного файла. Для папки возвращает общий размер всех файлов и подкаталогов, содержащихся в указанной папке. Синтаксис:

```
Объект.Size
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **File** или **Folder**.

В следующем примере показано использование свойства **Size** с объектом **Folder**:

```
Function ShowFolderSize(filespec)
  Dim fso, f, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFolder(filespec)
  s = UCase(f.Name) & " занимает " & f.size & " байт."
  ShowFolderSize = s
End Function
```

ПРИМЕЧАНИЕ
Если папка содержит большое количество файлов и/или имеет большой размер, то выполнение сценария может занять продолжительное время.

Свойство Type

Возвращает информацию о типе файла или папки. Например, для файла с расширением .TXT функция вернёт значение «Text Document» (в русифицированных операционных системах Windows результат на русском языке, например, «Текстовый документ»). Синтаксис:

```
Объект.Type
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **File** или **Folder**.

В следующем примере показано использование свойства **Type** с объектом **Folder**.

```
Function ShowFolderType(filespec)
  Dim fso, f, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFolder(filespec)
  s = UCase(f.Name) & " - это " & f.Type
  ShowFolderType = s
End Function
```

5.7.6.2. Методы объекта File

Метод Copy

Метод копирует указанный файл или папку из одного места в другое. Синтаксис:

```
Объект.Copy Приёмник[, Перезапись]
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **File** или **Folder**.
Приёмник	Обязательный элемент. Место, куда нужно скопировать файл или папку. Использование шаблонов (масок) НЕ допускается.
Перезапись	Не обязательный параметр. Логическое значение, которое определяет поведение метода, если файл, указанный в параметре **Приёмник**, уже существует. Если равно True, то файл будет перезаписан, если равно False, то копирование не выполнится. По умолчанию равно True.

ПРИМЕЧАНИЕ

Результаты, возвращаемые методом **Copy** объектов **File** или **Folder**, аналогичны результатам, получаемым при выполнении операций с использованием методов **FileSystemObject.CopyFile** или **FileSystemObject.CopyFolder**, где файл или папка связаны с параметром **Объект**, который передаётся в метод в качестве аргумента. Однако следует учесть, что эти методы более подходят для копирования нескольких файлов или папок.

В следующем примере показано использование метода **Copy**.

```
Dim fso, MyFile
Set fso = CreateObject("Scripting.FileSystemObject")
Set MyFile = fso.CreateTextFile("testfile-1.txt", True)
MyFile.WriteLine("Это проверка.")
MyFile.Close
Set MyFile = fso.GetFile("testfile-1.txt")
MyFile.Copy ("testfile-2.txt")
```

Метод Delete

Метод удаляет указанный файл или папку. Синтаксис:

```
Объект.Delete Форсировать
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **File** или **Folder**.
Форсировать	Не обязательный параметр. Логическое значение, которое определяет поведение метода. Если файл или папка, связанная с объектом, имеет атрибут **Только чтение**, то он всё равно будет удалён, если данный параметр имеет значение True. Иначе удаление не будет выполнено. По умолчанию равно False.

ПРИМЕЧАНИЕ

Если файл или папка не существует, то произойдёт ошибка. Метод **Delete** не различает пустые папки и папки с каким-либо содержимым. Указанная папка будет удалена, не зависимо от того, есть в ней какие-то файлы или нет.

Результаты, возвращаемые методом **Delete** объектов **File** или **Folder**, аналогичны результатам, получаемым при выполнении операций с использованием методов **FileSystemObject. DeleteFile** или **FileSystemObject. DeleteFolder**.

В следующем примере показано использование метода **Delete**.

```
Dim fso, MyFile
Set fso = CreateObject("Scripting.FileSystemObject")
Set MyFile = fso.CreateTextFile("testfile-1.txt", True)
MyFile.WriteLine("Это проверка.")
MyFile.Close
Set MyFile = fso.GetFile("testfile-1.txt")
MyFile.Delete
```

Метод Move

Метод перемещает указанный файл или папку из одного места в другое. Синтаксис:

```
Объект.Move Приёмник
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **File** или **Folder**.
Приёмник	Обязательный элемент. Место, куда нужно переместить файл или папку. Использование шаблонов (масок) НЕ допускается.

ПРИМЕЧАНИЕ

Результаты, возвращаемые методом **Move** объектов **File** или **Folder**, аналогичны результатам, получаемым при выполнении операций с использованием методов **FileSystemObject.MoveFile** или **FileSystemObject.MoveFolder**. Однако следует учесть, что эти методы более подходят для перемещения нескольких файлов или папок.

В следующем примере показано использование метода **Move**.

```
Dim fso, MyFile
Set fso = CreateObject("Scripting.FileSystemObject")
Set MyFile = fso.CreateTextFile("testfile-1.txt", True)
MyFile.WriteLine("Это проверка.")
MyFile.Close
Set MyFile = fso.GetFile("testfile-1.txt")
MyFile.Move "c:\"
```

Метод OpenAsTextStream

Метод открывает указанный файл и возвращает ссылку на объект **TextStream**, который может быть использован для чтения, записи и добавления данных в файл. Синтаксис:

```
Объект.OpenAsTextStream ([Режим, [Формат]])
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **File**.
Режим	Не обязательный элемент. Указывает режим ввода/вывода. Может быть одной из трёх констант: **ForReading**, **ForWriting** или **ForAppending**.
Формат	Не обязательный элемент. Одно из значений **Tristate**, которое определяет кодировку открытого файла. Если отсутствует, то файл открывается как ASCII-файл.

Параметры

Параметр **Режим** может принимать одно из следующих значений:

Константа	Значение	Описание
ForReading	1	Открыть файл только для чтения. Вы не сможете записать данные в этот файл.
ForWriting	2	Открыть файл для записи. Если файл с таким именем существует, он будет перезаписан, то есть все данные в этом файле будут потеряны.
ForAppending	8	Открыть файл для добавления данных в конец файла.

Параметр **Формат** может принимать одно из следующих значений:

Константа	Значение	Описание
TristateUseDefault	-2	Открытый файл будет использовать системные настройки по умолчанию.
TristateTrue	-1	Открыть файл как файл Unicode.
TristateFalse	0	Открыть файл как ASCII-файл.

ПРИМЕЧАНИЕ

Метод **OpenAsTextStream** обладает той же функциональностью, что и метод **OpenTextFile** объекта **FileSystemObject**. В дополнение метод **OpenAsTextStream** можно использовать для записи в файл.

В следующем примере показано использование метода **OpenAsTextStream**.

```
Function TextStreamTest
  Const ForReading = 1, ForWriting = 2, ForAppending = 8
  Const TristateUseDefault = -2, TristateTrue = -1, TristateFalse = 0
  Dim fso, f, ts
  Set fso = CreateObject("Scripting.FileSystemObject")
  fso.CreateTextFile "test-1.txt"            ' Создать файл.
  Set f = fso.GetFile("test-1.txt")
  Set ts = f.OpenAsTextStream(ForWriting, TristateUseDefault)
  ts.Write "Привет, МИР!"
  ts.Close
  Set ts = f.OpenAsTextStream(ForReading, TristateUseDefault)
  TextStreamTest = ts.ReadLine
  ts.Close
End Function
```

5.7.7. Объект TextStream

Объект предоставляет последовательный доступ к файлу.

ПРИМЕЧАНИЕ
В следующем кода **a** – это объект **TextStream**, возвращаемый методом **CreateTextFile** объекта **FileSystemObject**:

```
Dim fso, MyFile
Set fso = CreateObject("Scripting.FileSystemObject")
Set MyFile= fso.CreateTextFile("testfile.txt", True)
MyFile.WriteLine("Это проверка.")
MyFile.Close
```

WriteLine и **Close** – это методы объекта **TextStream**.

5.7.7.1. Свойства объекта TextStream

Свойство AtEndOfLine

Возвращает TRUE, если файловый указатель находится непосредственно перед маркером конца строки, иначе возвращает FALSE. Синтаксис:

```
Объект.AtEndOfLine
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **TextStream**.

Свойство **AtEndOfLine** применяется только для файлов, открытых для чтения, иначе произойдёт ошибка. В следующем примере показано использование свойства **AtEndOfLine**.

```
Function ReadEntireFile(filespec)
  Const ForReading = 1
  Dim fso, theFile, retstring
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set theFile = fso.OpenTextFile(filespec, ForReading, False)
  Do While theFile.AtEndOfLine <> True
    retstring = theFile.Read(1)
  Loop
  theFile.Close
  ReadEntireFile = retstring
End Function
```

Свойство AtEndOfStream

Возвращает TRUE, если файловый указатель находится в конце файла, иначе возвращает FALSE. Синтаксис:

```
Объект.AtEndOfStream
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **TextStream**.

Свойство **AtEndOfStream** применяется только для файлов, открытых для чтения, иначе произойдёт ошибка. В следующем примере показано использование свойства **AtEndOfStream**.

```
Function ReadEntireFileEOS(filespec)
  Const ForReading = 1
  Dim fso, theFile, retstring
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set theFile = fso.OpenTextFile(filespec, ForReading, False)
  Do While theFile.AtEndOfStream <> True
    retstring = theFile.ReadLine
  Loop
  theFile.Close
  ReadEntireFileEOS = retstring
End Function
```

Свойство Column

Возвращает номер столбца (колонки) текущей позиции символа в файле. Только чтение. Синтаксис:

```
Объект.Column
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **TextStream**.

ПРИМЕЧАНИЕ
После записи символа новой строки, но перед записью любого другого символа в эту строку свойство **Column** равно 1.

В следующем примере показано использование свойства **Column**.

```
Function GetColumn
  Const ForReading = 1, ForWriting = 2
  Dim fso, f, m
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
  f.Write "Hello world!"
  f.Close
  Set f = fso.OpenTextFile("c:\testfile.txt", ForReading)
  m =  f.ReadLine
  GetColumn = f.Column
End Function
```

Свойство Line

Возвращает номер текущей строки в файле. Только чтение. Синтаксис:

```
Объект.Line
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **TextStream**.

ПРИМЕЧАНИЕ
После инициализации и открытия файла и перед записью любой информации в файл свойство **Line** равно 1.

В следующем примере показано использование свойства **Line**.

```
Function GetLine
  Const ForReading = 1, ForWriting = 2
  Dim fso, f, ra
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
  f.Write "Привет, МИР!" & vbCrLf & "VB Script - это круто!" & vbCrLf
  Set f = fso.OpenTextFile("c:\testfile.txt", ForReading)
  ra =  f.ReadAll
  GetLine = f.Line
End Function
```

5.7.7.2. Методы объекта TextStream

Метод Close

Закрывает открытый файл **TextStream**. Синтаксис:

```
Объект.Close
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **TextStream**.

ПРИМЕЧАНИЕ

В следующем примере показано использование метода **Close**.

```
Sub CreateAFile
  Dim fso, MyFile
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set MyFile = fso.CreateTextFile("testfile.txt", True)
  MyFile.WriteLine("Это проверка.")
  MyFile.Close
End Sub
```

Метод Read

Читает указанное количество символов из файла **TextStream** и возвращает строку с результатом. Синтаксис:

```
Объект.Read(Символы)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **TextStream**.
Символы	Обязательный элемент. Количество символов, которые вы хотите прочитать из файла.

ПРИМЕЧАНИЕ

В следующем примере показано использование метода **Read**.

```
Function ReadTextFileTest
  Const ForReading = 1, ForWriting = 2, ForAppending = 8
  Dim fso, f, Msg
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.OpenTextFile("testfile.txt", ForWriting, True)
  f.Write "Привет, МИР!"
  Set f = fso.OpenTextFile("testfile.txt", ForReading)
  ReadTextFileTest = f.Read(6)
End Function
```

Метод ReadAll

Читает все символы из файла **TextStream** и возвращает строку с результатом. Синтаксис:

```
Объект.ReadAll
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **TextStream**.

ПРИМЕЧАНИЕ

Для больших файлов использование метода **ReadAll** приведёт к расходам ресурсов памяти. В таких случаях должны использоваться другие техники чтения файлов, такие как построчное чтение. В следующем примере показано использование метода **ReadAll**.

```
Function ReadAllTextFile
  Const ForReading = 1, ForWriting = 2, ForAppending = 8
  Dim fso, f, Msg
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.OpenTextFile("testfile.txt", ForWriting, True)
  f.Write "Привет, МИР!"
  Set f = fso.OpenTextFile("testfile.txt", ForReading)
  ReadAllTextFile =  f.ReadAll
End Function
```

Метод ReadLine

Читает строку (кроме символа конца строки) из файла **TextStream** и возвращает строку с результатом. Синтаксис:

```
Объект.ReadLine
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **TextStream**.

В следующем примере показано использование метода **ReadLine**.

```
Function ReadLineTextFile
  Const ForReading = 1, ForWriting = 2
  Dim fso, MyFile
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set MyFile = fso.OpenTextFile("testfile.txt", ForWriting, True)
  MyFile.WriteLine "Привет, МИР!"
  MyFile.WriteLine "Быстрый рыжий лис"
  MyFile.Close
  Set MyFile = fso.OpenTextFile("testfile.txt", ForReading)
  ReadLineTextFile = MyFile.ReadLine  ' Возвращает "Привет, МИР!"
End Function
```

Метод Skip

Пропускает указанное количество символов при чтении из файла **TextStream**. Синтаксис:

```
Объект.Skip(Символы)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **TextStream**.
Символы	Обязательный элемент. Количество символов, которые вы хотите пропустить при чтении из файла.

Пропускаемые символы отбрасываются. В следующем примере показано использование метода **Skip**. Здесь будут пропущены первые 8 символов текущей при чтении этой строки (то есть прочитано будет только слово «МИР!»).

```
Function SkipTextFile
  Const ForReading = 1, ForWriting = 2
  Dim fso, f
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.OpenTextFile("testfile.txt", ForWriting, True)
  f.Write "Привет, МИР!"
  Set f = fso.OpenTextFile("testfile.txt", ForReading)
  f.Skip(8)
  SkipTextFile =  f.ReadLine
End Function
```

Метод SkipLine

Пропускает следующую строку при чтении из файла **TextStream**. Синтаксис:

```
Объект.SkipLine
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **TextStream**.

Пропуск строки означает, что отбрасываются все символы в данной строке, включая символ конца строки. Если файл не открыт для чтения, происходит ошибка. В следующем примере показано использование метода **SkipLine**.

```
Function SkipLineInFile
  Const ForReading = 1, ForWriting = 2
  Dim fso, f
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.OpenTextFile("testfile.txt", ForWriting, True)
  f.Write "Привет, МИР!" & vbCrLf & "VB Script - это круто!" & vbCrLf
  Set f = fso.OpenTextFile("testfile.txt", ForReading)
  f.SkipLine
  SkipLineInFile = f.ReadLine
End Function
```

Метод Write

Записывает указанную строку в файл **TextStream**. Синтаксис:

```
Объект.Write(Строка)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **TextStream**.
Строка	Обязательный элемент. Текст, который вы хотите записать в файл.

Указанная строка записывается в файл без промежуточных пробелов или символов между каждой строкой. Используйте метод **WriteLine** для записи символа новой строки или строки, которая заканчивается символом новой строки. В следующем примере показано использование метода **Write**.

```
Function WriteToFile
  Const ForReading = 1, ForWriting = 2
  Dim fso, f
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.OpenTextFile("testfile.txt", ForWriting, True)
  f.Write "Привет, МИР!"
  Set f = fso.OpenTextFile("testfile.txt", ForReading)
  WriteToFile =  f.ReadLine
End Function
```

Метод WriteLine

Записывает указанную строку с символом конца строки в файл **TextStream**. Синтаксис:

```
Объект.WriteLine([Строка])
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **TextStream**.
Строка	Не обязательный элемент. Текст, который вы хотите записать в файл. Если отсутствует, то в файл записывается символ новой строки.

В следующем примере показано использование метода **WriteLine**.

```
Function WriteLineToFile
  Const ForReading = 1, ForWriting = 2
  Dim fso, f
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.OpenTextFile("testfile.txt", ForWriting, True)
  f.WriteLine "Привет, МИР!"
  f.WriteLine "VB Script - это круто!"
  Set f = fso.OpenTextFile("testfile.txt", ForReading)
  WriteLineToFile = f.ReadAll
End Function
```

Метод WriteBlankLines

Записывает указанное количество символов конца строки в файл **TextStream**. Синтаксис:

```
Объект.WriteBlankLines(Строки)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **TextStream**.
Строки	Обязательный элемент. Количество символов новой строки, которые вы хотите записать в файл.

В следующем примере показано использование метода **WriteBlankLines**.

```
Function WriteBlankLinesToFile
  Const ForReading = 1, ForWriting = 2
  Dim fso, f
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.OpenTextFile("testfile.txt", ForWriting, True)
  f.WriteBlankLines 2
  f.WriteLine "Привет, МИР!"
  Set f = fso.OpenTextFile("testfile.txt", ForReading)
  WriteBlankLinesToFile = f.ReadAll
End Function
```

5.7.8. Объект Match

Объект предоставляет доступ к свойствам найденных совпадений регулярных выражений. Свойства имеют атрибут «Только чтение».

ПРИМЕЧАНИЕ

Объект **Match** может быть создан только с помощью метода **Execute** объекта **RegExp**, который на самом деле возвращает коллекцию объектов **Match**. Все свойства объекта **Match** имеют атрибут «Только чтение».

Если регулярное выражение выполнено, то результатом может быть ноль или больше объектов **Match**. Каждый объект **Match** предоставляет доступ к строке, найденной при помощи регулярного выражения, к длинне строки и индексу, где было найдено совпадение.

В следующем примере показано использование метода **Match**:

```
Function RegExpTest(patrn, strng)
  Dim regEx, Match, Matches         ' Создать переменную.
  Set regEx = New RegExp            ' Создать регулярное выражение.
  regEx.Pattern = patrn             ' Установить шаблон.
  regEx.IgnoreCase = True           ' Установить нечувствительность к регистру.
  regEx.Global = True               ' Установить глобальную применимость.
  Set Matches = regEx.Execute(strng)      ' Выполнить поиск.
  For Each Match in Matches         ' Перебрать коллекцию объектов Match.
    RetStr = RetStr & "Match " & I & " найден на позиции "
    RetStr = RetStr & Match.FirstIndex & ". Значение Match = "'
    RetStr = RetStr & Match.Value & "'." & vbCRLF
  Next
  RegExpTest = RetStr
End Function

MsgBox(RegExpTest("is.", "IS1 is2 IS3 is4"))
```

5.7.8.1. Свойства объекта Match

Свойство FirstIndex

Возвращает позицию в строке поиска где найдено совпадение. Синтаксис:

```
Объект.FirstIndex
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Match**.

Свойство **FirstIndex** использует смещение (основание – ноль) от начала строки поиска. В других словах первый символ в строке идентифицируется как символ ноль (0). В следующем примере показано использование свойства **FirstIndex**.

```
Function RegExpTest(patrn, strng)
  Dim regEx, Match, Matches          ' Создать переменную.
  Set regEx = New RegExp             ' Создать регулярное выражение.
  regEx.Pattern = patrn              ' Установить шаблон.
  regEx.IgnoreCase = True            ' Установить нечувствительность к регистру.
  regEx.Global = True                ' Установить глобальную применимость.
  Set Matches = regEx.Execute(strng)      ' Выполнить поиск.
  For Each Match in Matches          ' Перебрать коллекцию объектов Match.
    RetStr = RetStr & "Match " & I & " найден на позиции "
    RetStr = RetStr & Match.FirstIndex & ". Значение Match = "
    RetStr = RetStr & Match.Value & "'." & vbCRLF
  Next
  RegExpTest = RetStr
End Function

MsgBox(RegExpTest("is.", "IS1 is2 IS3 is4"))
```

Здесь мы передаём в функцию RegExpTest маску (шаблон) поиска **is.** и строку **IS1 is2 IS3 is4**, где будет выполняться поиск. Предварительно мы устанавливаем нечувствительность к регистру и флаг поиска всех совпадений. То есть в результете выполнения данной функции будет возвращено четыре объекта **Match**, так как в переданной строке текст **is** встречается 4 раза (без учёта регистра).

Свойство Length

Возвращает длину найденного совпадения в строке поиска. Синтаксис:

```
Объект.Length
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Match**.

В следующем примере показано использование свойства **Length**.

```
Function RegExpTest(patrn, strng)
  Dim regEx, Match, Matches          ' Создать переменную.
  Set regEx = New RegExp             ' Создать регулярное выражение.
  regEx.Pattern = patrn              ' Установить шаблон.
  regEx.IgnoreCase = True            ' Установить нечувствительность к регистру.
  regEx.Global = True                ' Установить глобальную применимость.
  Set Matches = regEx.Execute(strng)      ' Выполнить поиск.
  For Each Match in Matches          ' Перебрать коллекцию объектов Match.
    RetStr = RetStr & "Match " & I & " найден на позиции "
    RetStr = RetStr & Match.FirstIndex & ". Длина Match = "'
    RetStr = RetStr & Match.Length
    RetStr = RetStr & " символов." & vbCRLF
  Next
  RegExpTest = RetStr
End Function

MsgBox(RegExpTest("is.", "IS1 is2 IS3 is4"))
```

Свойство Value

Возвращает значение или текст совпадения, найденного в строке поиска. Синтаксис:

```
Объект.Value
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **Match**.

В следующем примере показано использование свойства **Value**.

```
Function RegExpTest(patrn, strng)
  Dim regEx, Match, Matches          ' Создать переменную.
  Set regEx = New RegExp             ' Создать регулярное выражение.
  regEx.Pattern = patrn              ' Установить шаблон.
  regEx.IgnoreCase = True            ' Установить нечувствительность к регистру.
  regEx.Global = True                ' Установить глобальную применимость.
  Set Matches = regEx.Execute(strng)      ' Выполнить поиск.
  For Each Match in Matches          ' Перебрать коллекцию объектов Match.
    RetStr = RetStr & "Match " & I & " найден на позиции "
    RetStr = RetStr & Match.FirstIndex & ". Значение Match = "
    RetStr = RetStr & Match.Value & "'." & vbCRLF
  Next
  RegExpTest = RetStr
End Function

MsgBox(RegExpTest("is.", "IS1 is2 IS3 is4"))
```

5.7.9. Объект RegExp

Объект предоставляет поддержку простых регулярных выражений.

В следующем примере показано использование свойства **RegExp**.

```
Function RegExpTest(patrn, strng)
  Dim regEx, Match, Matches         ' Создать переменную.
  Set regEx = New RegExp             ' Создать регулярное выражение.
  regEx.Pattern = patrn              ' Установить шаблон.
  regEx.IgnoreCase = True            ' Установить нечувствительность к регистру.
  regEx.Global = True                ' Установить глобальную применимость.
  Set Matches = regEx.Execute(strng)      ' Выполнить поиск.
  For Each Match in Matches          ' Перебрать коллекцию объектов Match.
    RetStr = RetStr & "Match " & I & " найден на позиции "
    RetStr = RetStr & Match.FirstIndex & ". Значение Match = "
    RetStr = RetStr & Match.Value & "'." & vbCRLF
  Next
  RegExpTest = RetStr
End Function

MsgBox(RegExpTest("is.", "IS1 is2 IS3 is4"))
```

5.7.9.1. Свойства объекта RegExp

Свойство Global

Устанавливает или возвращает логическое значение, которое определяет, нужно ли учитывать все совпадения во входной строке поиска или только первое. Синтаксис:

```
Объект.Global [= True | False ]
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **RegExp**. Значение свойства **Global** равно TRUE, если поиск применяется для всей строки, иначе равно FALSE. По умолчанию равно TRUE.

В следующем примере показано использование свойства **Global**. Чтобы увидеть эффект от использования этого свойства, измените его значение.

```
Function RegExpTest(patrn, strng)
  Dim regEx, Match, Matches         ' Создать переменную.
  Set regEx = New RegExp            ' Создать регулярное выражение.
  regEx.Pattern = patrn             ' Установить шаблон.
  regEx.IgnoreCase = True           ' Установить нечувствительность к регистру.
  regEx.Global = True               ' Установить глобальную применимость.
  Set Matches = regEx.Execute(strng)      ' Выполнить поиск.
  RegExpTest = RetStr
End Function

MsgBox(RegExpTest("is.", "IS1 is2 IS3 is4"))
```

Свойство IgnoreCase

Устанавливает или возвращает логическое значение, которое определяет, нужно ли учитывать регистр символов при поиске или нет. Синтаксис:

```
Объект.IgnoreCase [= True | False ]
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **RegExp**. Значение свойства **IgnoreCase** равно TRUE, если при поиске НЕ нужно учитывать регистр символов, иначе равно FALSE. По умолчанию равно TRUE.

В следующем примере показано использование свойства **IgnoreCase**. Чтобы увидеть эффект от использования этого свойства, измените его значение.

```
Function RegExpTest(patrn, strng)
  Dim regEx, Match, Matches         ' Создать переменную.
  Set regEx = New RegExp            ' Создать регулярное выражение.
  regEx.Pattern = patrn             ' Установить шаблон.
  regEx.IgnoreCase = True           ' Установить нечувствительность к регистру.
  Set Matches = regEx.Execute(strng)      ' Выполнить поиск.
  RegExpTest = RetStr
End Function

MsgBox(RegExpTest("is.", "IS1 is2 IS3 is4"))
```

Свойство Pattern

Устанавливает или возвращает шаблон (маску) реуглярного выражения, который будет используется для поиска. Синтаксис:

```
Объект.Pattern [= "СтрокаПоиска" ]
```

Свойство имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **RegExp**.
СтрокаПоиска	Не обязательный параметр. Регулярное строковое выражение для поиска. Может включать любые символы регулярных выражений, описанные в таблице раздела «Настройки».

Настройки

Специальные символы и последовательности используются при формировании шаблонов регулярных выражений. В следующей таблице приводятся описание и примеры символов и последовательностей, которые могут использоваться.

Символ	Описание
****	Отмечает следующий символ как специальный символ или букву. Например, «n» соответствует символу «n», в то время как «\n» соответствует символу новой строки. Последовательность «\\» соответствует символу «\», последовательность «/(» соответствует символу «(».
^	Соответствует началу ввода.
$	Соответствует концу ввода.
*****	Соответствует появлению предыдущего символа ноль или более раз. Например, «zo*» будет соответствовать как «z», так и «zoo».
+	Соответствует появлению предыдущего символа один или более раз. Например, «zo+» будет соответствовать «zoo», но не будет соответствовать «z».
?	Соответствует появлению предыдущего символа ноль или один раз. Например, «a?ve?» будет соответствовать «ve» в «never».
.	Соответствует любому одиночному символу, кроме символа новой строки.
(pattern)	Соответствует **pattern** и запоминает соответствие. Соответствующая подстрока может быть получена из результирующей коллекции **Matches**, используя **Item [0]...[n]**. Для поиска совпадений круглых скобок используйте «\(» или «\)».
x**|**y	Соответствует **x** или **y**. Например, «z|food» соответствует «z» или «food». «(z|f)ood» соответствует «zoo» или «food».
{n**}**	**n** – это не отрицательное целое число. Точное соответствие n раз. Например, «o{2}» не соответствет «o» в «Bob», но соответствует первым двум «o» в слове «foooood».
{n**,}**	**n** – это не отрицательное целое число. Соответствие менее n раз. Например, «o{2,}» не соответствет «o» в «Bob», но соответствует всем «o» в слове «foooood». «o{1,}» эквивалентно «o+». «o{0,}» эквивалентно «o*».
{n**,**m**}**	**n** и **m** – это не отрицательное целое число. Соответствие менее n раз и не более m раз. Например, «o{1,3}» первым трём «o» в слове «foooood». «o{0,1}» эквивалентно «o?».
[xyz**]**	Множество символов. Соответствует любому из указанных символов. Например, «[abc]» соответствует «a» в «plain».
[^xyz**]**	Отрицательное множество символов. Соответствует любому из НЕ указанных символов. Например, «[^abc]» соответствует «p» в «plain».
[a-z**]**	Диапазон символов. Соответствует всем символам в указанном диапазоне. Например, «[a-z]» соответствует любой букве алфавита в нижнем регистре в диапазоне от «a» до «z».

Символ	Описание
[^m-z**]**	Отрицательное диапазон символов. Соответствует любому, НЕ входящему в указанный диапазон. Например, «[^m-z]» соответствует любому символу (например, «a»), не входящему в диапазон от «m» до «z».
\b	Соответствие окончанию слова, то есть позиции между последней буквой слова и пробелом. Например, «er\b» соответствует «er» в «never», но НЕ соответствует «er» в «verb».
\B	Соответствие началу слова. Например, «ea*r\B» соответствует «ear» в «never early».
\d	Соответствует символу-цифре. Эквивалентно [0-9].
\D	Соответствует НЕ цифровомц символу. Эквивалентно [^0-9].
\f	Соответствует символу перехода к следующей странице.
\n	Соответствует символу новой строки.
\r	Соответствует символу возврата каретки.
\s	Соответстсвует любым «пустым» символам: пробел, табуляция, переход к новой странице и т.п. Эквивалентно «[\f\n\r\t\v]».
\S	Соответстсвует любым НЕ «пустым» символам. Эквивалентно «[^ \f\n\r\t\v]».
\t	Соответствует символу табуляции.
\v	Соответствует символу вертикальной табуляции.
\w	Соответствует любому текстовому символу, включая символ подчёркивания. Эквивалентно «[A-Za-z0-9_]».
\W	Соответствует любому НЕ текстовому символу, включая символ подчёркивания. Эквивалентно «[^A-Za-z0-9_]».
\num	Соответствует num, где num – положительное целое число. Ссылается назад к запомненным совпадениям. Например, «(.)\1» совпадает с двумя следующими друг за другом одинаковыми символами.
\n	Соответствует n, где n – восьмеричное значение. Восьмеричное число должно быть длинной в 1, 2 или 3 цифры. Например, оба числа «\11» и «\011» соответствуют символу табуляции. «\0011» эквивалентно «\001» & «1». Восьмеричные значения должны быть не более 256. Если они больше, то только первые две цифры используются в выражении. В регулярных выражениях допускается использовать ASCII-коды.
\xn	Соответствует n, где n – шестнадцатиричное значение. Восьмеричное число должно быть длинной в 2 цифры. Например, «\x41» соответствует «A». «\041» эквивалентно «\04» & «1». В регулярных выражениях допускается использовать ASCII-коды.

В следующем примере показано использование свойства **Pattern**.

```
Function RegExpTest(patrn, strng)
  Dim regEx, Match, Matches         ' Создать переменную.
  Set regEx = New RegExp            ' Создать регулярное выражение.
  regEx.Pattern = patrn             ' Установить шаблон.
  regEx.IgnoreCase = True           ' Установить нечувствительность к регистру.
  Set Matches = regEx.Execute(strng)      ' Выполнить поиск.
  RegExpTest = RetStr
End Function

MsgBox(RegExpTest("is.", "IS1 is2 IS3 is4"))
```

5.7.9.2. Методы объекта RegExp

Метод Execute

Выполняет поиск регулярного выражения для указанной строки. Синтаксис:

```
Объект.Execute(Строка)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **RegExp**.
Строка	Обязательный элемент. Текстовая строка, для которой выполняется регулярное выражение.

ПРИМЕЧАНИЯ

Шаблон (маска) регулярного выражения устанавливается через свойство **Pattern** объекта **RegExp**.

Метод **Execute** возвращает коллекцию **Matches**, содержащую объект **Match** для каждого совпадения, найденного в указанной **Строке**. Если совпадений не найдено, то метод **Execute** возвращает пустую коллекцию **Matches**.

В следующем примере показано использование метода **Execute**.

```
Function RegExpTest(patrn, strng)
  Dim regEx, Match, Matches         ' Создать переменную.
  Set regEx = New RegExp            ' Создать регулярное выражение.
  regEx.Pattern = patrn             ' Установить шаблон.
  regEx.IgnoreCase = True           ' Установить нечувствительность к регистру.
  Set Matches = regEx.Execute(strng)      ' Выполнить поиск.
  RegExpTest = RetStr
End Function

MsgBox(RegExpTest("is.", "IS1 is2 IS3 is4"))
```

Метод Replace

Заменяет текст, найденный в регулярном выражении. Синтаксис:

```
Объект.Replace(Строка1, Строка2)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **RegExp**.
Строка1	Обязательный элемент. Текстовая строка, в которой будет заменён текст, соответствующий регулярному выражению.
Строка2	Обязательный элемент. Текстовая строка, в которой будет заменён текст, соответствующий регулярному выражению.

ПРИМЕЧАНИЯ

Шаблон (маска) регулярного выражения устанавливается через свойство **Pattern** объекта **RegExp**.

Метод **Replace** возвращает копию **Строки1** с текстом **RegExp.Pattern**, заменённым **Строкой2**. Если совпадений не найдено, то копия **Строки1** не изменяется.

В следующем примере показано использование метода **Replace**.

```
Function ReplaceTest(patrn, replStr)
  Dim regEx, str1                   ' Создать переменные.
  str1 = "Быстрый рыжий лис перепрыгнул через ленивого пса."
  Set regEx = New RegExp            ' Создать регулярное выражение.
  regEx.Pattern = patrn             ' Установить шаблон.
  regEx.Global = True               ' Установить глобальную применимость.
  regEx.IgnoreCase = True           ' Установить нечувствительность к регистру.
  ReplaceTest = regEx.Replace(str1, replStr)    ' Выполнить замену.
End Function

MsgBox(ReplaceTest("лис", "кот"))   ' Заменяет «лис» на «кот».
```

Кроме этого метод **Replace** может заменять подвыражения в шаблоне. В следующем примере вызов предыдущей функции приведёт к тому, что каждая пара слов в тексте будет поменяна местами.

```
MsgBox(ReplaceTest("(\S+)(\s+)(\S+)", "$3$2$1"))
```

То есть эта функция вернёт следующий результат:

```
рыжий Быстрый перепрыгнул лис ленивого через пса.
```

Метод Test

Выполняет поиск регулярного выражения для указанной строки и возвращает логическое значение, которое сообщает о том, найдено совпадение или нет. Синтаксис:

```
Объект.Test(Строка)
```

Метод имеет следующие составляющие:

Элемент	Описание
Объект	Обязательный элемент. Всегда имя объекта **RegExp**.
Строка	Обязательный элемент. Текстовая строка, для которой выполняется регулярное выражение.

ПРИМЕЧАНИЯ

Шаблон (маска) регулярного выражения устанавливается через свойство **Pattern** объекта **RegExp**. Значение свойства **RegExp.Global** не влияет на результат.

Метод **Test** возвращает TRUE, если найдено хотя бы одно совпадение с шаблоном. Если совпадений не найдено, метод возвращает FALSE.

В следующем примере показано использование метода **Test**.

```
Function RegExpMetTest(patrn, strng)
  Dim regEx, retVal                  ' Создать переменные.
  Set regEx = New RegExp             ' Создать регулярное выражение.
  regEx.Pattern = patrn              ' Установить шаблон.
  regEx.IgnoreCase = False           ' Установить чувствительность к регистру.
  retVal = regEx.Test(strng)         ' Выполнить попытку поиска.
  If retVal Then
    RegExpMetTest = "Найдено одно или более совпадений."
  Else
    RegExpMetTest = "Совпадений не найдено."
  End If
End Function

MsgBox(RegExpMetTest("is.", "IS1 is2 IS3 is4"))
```

ОБ АВТОРЕ

На всякий случай представлюсь (вдруг кому интересно).

Это я – Поляков Андрей Валерьевич (можно просто Андрей))).

По образованию я инженер. Работаю руководителем инженерного отдела в крупном (по меркам нашего региона) агрохолдинге.

Это основная работа.

Но есть у меня и другие интересы. Я пишу книги, создаю обучающие курсы. А также немного (на уровне любопытства) занимаюсь инфо-бизнесом.

Кроме того у меня есть несколько сайтов с полезной информацией.

Итак, вот список ссылок на мои сайты, которые (во всяком случае, я на это надеюсь) могут оказаться Вам полезными.

Сначала личное:

Мой блог: http://av-inf.blogspot.ru

В контакте: http://vk.com/id185471101

Фейсбук: https://www.facebook.com/100008480927503

LinkedIn: http://ru.linkedin.com/pub/%D0%B0%D0%BD%D0%B4%D1%80%D0%B5%D0%B9-%D0%BF%D0%BE%D0%BB%D1%8F%D0%BA%D0%BE%D0%B2/86/906/6b1/

Одноклассники: http://ok.ru/polakoff

Мой круг: http://polyakovandrey44.moikrug.ru/

Гугл Плюс: https://plus.google.com/104478558796835580670/posts

Может сложиться впечатление, что я не вылажу из социальных сетей))) На самом деле это не так – я там редкий гость. Однако учётные записи в разных сетях нужны мне для общения с читателями.

P.S. Возможно, на тот момент, когда вы получите эту книгу, что-то изменится. Актуальную информацию можно найти здесь: http://info-master.su/contact.php

Бесплатные книги и видеокурсы:

http://info-master.su/mb.php - список всех моих бесплатных рассылок, где вы можете получить мои бесплатные книги и другие материалы.

Теперь сайты и проекты:

Реальные отзывы об инфо-продуктах
Здесь вы найдёте реальные (а не купленные и не придуманные) отзывы о разных инфо-продуктах. Кстати, можете поделиться своими впечатлениями о книге, тренинге, видеокурсе и т.п. – я буду только рад. Ссылка: http://realnie-otzivi.blogspot.ru

Инфо-мастер
Мой основной сайт, где вы можете найти все мои книги (и не только мои):
http://info-master.su

Основы программирования
Подраздел основного сайта, посвящённый программированию:
http://info-master.su/programming/index.php

Психология машин
Мой новый проект. Пока всё только начинается. Но посмотреть уже можно:
https://www.facebook.com/817221088320114

Автогид
Это ну ваще новый проект (на начало 2015 года). Пока практически пустой, но развитие планируется: http://avto-gid.blogspot.ru/

Ближе к железу
Сайт о программировании на ассемблере: http://av-assembler.ru

Информация
Тоже мой сайт, содержащий разнообразную информацию – от книг и рефератов до статей об автомобилях: http://av-mag.ru

Системы навигации и мониторинга
Подраздел предыдущего сайта, полностью посвящённый системам навигации, мониторинга, а также логистике и связанным со всем этим темам:
http://av-mag.ru/snm/sistemy-navigacii-i-monitoringa.php

Физика для всех
Сайт о физике: http://av-physics.narod.ru

Всё для студента
Этот сайт был создан, когда я учился в университете и был старостой группы.
http://tz-5133.narod.ru

В помощь студенту и инженеру
А это мой самый первый сайт. Создан аж в 2003 году. Я его давно не поддерживаю. Но народ заходит, поэтому и не удаляю: http://avprog.narod.ru