АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Кафедра информационных систем в экономике

Основы объектной модели MS'Office
Использование VBA и Windows Script Host

Барнаул 2003

Составитель: к.ф.-м.н., доцент кафедры ИСЭ АГУ
Юдинцев Алексей Юрьевич

Напечатано в типографии экономического факультета АГУ
пр. Социалистический, 68, тел. 36-42-32

Содержание

Объектные модели MS'Word и MS'Excel

Объектная модель MS'Word

Объектная модель MS'Word состоит из большого числа объектов и коллекций – массивов объектов. Как каждая коллекция, так и каждый объект, в свою очередь, обладают набором собственных свойств, событий и методов. Работают с коллекциями, так же как и с объектами. Обращение же к объекту – элементу коллекции происходит с использованием индекса, который указывает номер элемента коллекции.
В состав основных элементов объектной модели MS'Word входят объекты: application, activedocument, activewindow, selection, и коллекции – documents, paragraphs, sentences, words, characters, tables, shapes, inlineshapes …

Основное окно MS'Word представляет собой объект класса application. Создать новый экземпляр MS'Word из VFP можно командой:
Wrd= createobject("word.application")

Вновь созданный объект необходимо сделать видимым: wrd.visible = .t.
Статусом основного окна приложения можно управлять: wrd.windowstate = 1 – распахнуть окно, wrd.windowstate = 0 – вернуть окно в нормальный вид, wrd.windowstate = 2 – минимизировать окно.
Перемещать и менять размеры окна, можно используя свойства: top, left, height, width.
Wrd.top = 20
Wrd.left = 100
Wrd.height = 400
Wrd.width = 500
Закрыть приложение можно методом quit (SaveChanges, WdOriginalFormat, RouteDocument)

Необязательный параметр SaveChanges определяет, будут ли сохраняться изменения в документе.
SaveChanges = 0 – не сохранять изменения
SaveChanges = -2 – вывести запрос на сохранение документа
SaveChanges = -1 – сохранить документ

Необязательный параметр WdOriginalFormat определяет, будет ли производиться преобразование формата в случае если формат исходного документа отличался от формата Word. Он может принимать следующие значения
WdOriginalFormat = 1 – документ сохранится в оригинальном формате,
WdOriginalFormat = 2 – выведется запрос на преобразование формата документа
WdOriginalFormat = 0 – будет произведено преобразование формата документа к формату MS'Word.

Необязательный параметр RouteDocument может принимать значения .T. или .F. и предназначен для перенаправления документов другому пользователю посредством OutLook или других средств.

Объекты документов MS'Word находятся в коллекции documents.

Количество открытых документов можно получить, используя свойство count. Если открыть несколько документов, то команда ? wrd.documents.count выдаст на экран число открытых документов.
Метод add добавляет еще один элемент в коллекцию документов:
Команды
Wrd.documents.Add
? Wrd.documents.count
выведут на экран количество открытых документов.
Метод open предназначен для открытия документа. При открытии нужно задать обязательный параметр – название документа, остальные параметры являются дополнительными.
Wrd.documents.Open("u:\test.doc")

К открытому документу можно обратиться через индекс
? Wrd.documents(1).name
или по имени
? wrd.documents("test.doc").name

Коллекция paragraphs предназначена для работы с абзацами текста. Добавить параграф можно методом add.
wrd.documents(1).paragraphs.add
? Wrd.documents(1).paragraphs.count

Форматирование абзацев текста

Основные свойства абзаца:
Выравнивание (Alignment) текстового абзаца:
wrd.Documents(1).Paragraphs(1).Format.Alignment = 1 – по центру,
wrd.Documents(1).Paragraphs(2).Format.Alignment = 0 – по левому краю,
wrd.Documents(1).Paragraphs(3).Format.Alignment = 2 – по правому краю,
wrd.Documents(1).Paragraphs(4).Format.Alignment = 3 – по ширине.
Левая (LeftIndent) и правая (RightIndent) граница текста абзаца:
wrd.Documents(1).Paragraphs(1).Format.LeftIndent = 10
wrd.Documents(1).Paragraphs(1).Format.RightIndent = 30
Абзацный отступ (красная строка) абзаца (FirstLineIndent):
Wrd.documents(1).Paragraphs(1).Format.FirstLineIndent =45
Интервалы перед (SpaceBefore) и после (SpaceAfter) абзаца:
Wrd.documents(1).Paragraphs(1).Format.SpaceBefore = 12
Wrd.documents(1).Paragraphs(1).Format.SpaceAfter = 24
Межстрочный интервал (LineSpacingRule):
wrd.Documents(1).Paragraphs(1).Format.LineSpacingRule = 0.5 – одинарный интервал,
wrd.Documents(1).Paragraphs(1).Format.LineSpacingRule = 1 – полуторный интервал,
wrd.Documents(1).Paragraphs(1).Format.LineSpacingRule = 1.5 – двойной интервал.

Форматирование шрифта текста

Для форматирования шрифта текста используется объект font. Основные свойства объекта font: Name – название шрифта (текстовая строка), Size – размер шрифта (число), Bold – управление стилем «полужирный шрифт» (логическое значение), Italic – управление стилем «курсив» (логическое значение), Underline – управление стилем «подчеркнутый шрифт» (логическое значение), UnderlineColor – цвет подчеркивания (число, можно использовать макрос RGB(,,)), StrikeThrough – управление стилем «зачеркнутый текст» (логическое значение), DoubleStrikeThrough – управление стилем «текст зачеркнутый двойной линией»

(логическое значение), SmallCaps – перевод заглавных букв в прописные (логическое значение), AllCaps – перевод всех букв текста в заглавные (логическое значение), Color – цвет текста (число, можно использовать макрос RGB(,,)), Superscript – верхний индекс (логическое значение), Subscript – нижний индекс (логическое значение), Spacing – расстояние между буквами по горизонтали (текст в разрядку или сжатый текст) (число), Scaling – ширина букв (число), Position – смещение относительно базовой линии по вертикали (число).

Объект Font является свойством элементов базовых коллекций: characters, words, sentences. Так, например, можно изменить размеры шрифта первой буквы, первого слова и первого предложения нашего документа:
wrd.Documents(1).Characters(1).Font.Size=50
wrd.Documents(1).Words(1).Font.Size=35
wrd.Documents(1).Sentences(1).Font.Size=25

Как правило, наиболее часто свойства шрифта – свойства объекта font, задаются через промежуточный объект range – диапазон.
Занесем текста в первый абзац нашего документа, используя объект range:
Wrd.Documents(1).Paragraphs(1).Range.Text = "Пробная текстовая строка"
Поменяем размер шрифта, название шрифта и установим стиль – «полужирный»:
Wrd.Documents(1).Paragraphs(1).Range.Font.Size= 50
Wrd.Documents(1).Paragraphs(1).Range.Font.Name="Arial"
Wrd.Documents(1).Paragraphs(1).Range.Font.Bold = .T.

Работа с выделениями и диапазонами

Объект и свойство Range

Объект range – диапазон, представляет собой некоторую непрерывную область документа. Каждый объект Range определяется начальной и конечной позициями. Объект Range независим от выделения, так что вы легко можете оперировать диапазонами без изменения выделения. Также возможно объявить несколько диапазонов, в то время как выделение может быть в каждом конкретном случае только одно.
В следующем примере создается объект MS'Word, в него добавляется новый документ Word и в этом документе первые 10 символов объявляются диапазоном mR. Далее к этому диапазону применяется форматирование шрифта – изменяется размер и начертание шрифта:
Wrd = createobject("Word.Application")
Wrd.documents.add
mR = ActiveDocument.Range(Start:=0, End:=10)
mR.Font.Size = 26
mR.Font.Italic = .T.

Наряду с объектом Range, существует свойство Range, которое возвращает объект Range определяемый началом и концом объекта. Свойство **Range** есть у большого числа объектов (например, **ActiveDocument**, **Paragraph** ...).
Следующий пример возвращает объект **Range,** который соответствует первому параграфу активного документа

```
aRng = wrd.ActiveDocument.Paragraphs(1).Range

Далее установим в диапазоне жирный шрифт:
aRng.font.bold = .t.

Выровняем текст вправо:
aRng.paragraphformat.alignment = 2
```

Получим начальную позицию диапазона второго параграфа:
pos = wrd.ActiveDocument.Paragraphs(2).Range.Start
? pos
Следующий пример возвращает объект **Range** который возвращает диапазон, начинающийся с начала второго параграфа и заканчивающийся концом четвертого параграфа активного документа

```
aRng = wrd.ActiveDocument.Range( wrd.ActiveDocument.Paragraphs(2).Range.Start,
wrd.ActiveDocument.Paragraphs(4).Range.End)
```

arng.font.size=50
? arng.characters.count
? arng.words.count

Объект и свойство selection

Объект selection – выделение. Объект **Selection** представляет собой выделенную (подсвеченную) область в документе. В документе может быть только одна выделенная область.
Создайте объектную переменную wrd для приложения Word, добавьте или откройте документ, выделите в нем некоторую область, тогда команда ? wrd.selection.characters.count даст количество символов в выделении,
? wrd.selection.range.text
распечатает текст выделения.

Выделять можно в документе не только текст, но и другие объекты. Для определения типа выделения существует свойство type (для текста тип выделения – 2)
? wrd.selection.type
выведет на экран 2.
Ниже приведены некоторые наиболее часто употребляющиеся типы выделения:

Type	Описание объекта	Название константы WordBasic
1	Нет выделения	wdSelectionIP
2	Обычное выделение	wdSelectionNormal
4	Колонка	wdSelectionColumn
5	Строка	wdSelectionRow
7	Встроенный плавающий объект	wdSelectionInlineShape
8	Плавающий объект	wdSelectionShape

Нижеприведенный пример анализирует выделение и, если выделен текст, выводит его на экран.

```
If wrd.Selection.Type = 2
   ? wrd.selection.Text
else
   ?  "Выделен не текст, а другой объект"
endif
```

Следующий пример копирует выделение из одного окна в другое.

```
If wrd.Windows.Count >= 2
    Wrd.Windows(1).Selection.Copy
```

```
    Wrd.Windows(1).Next.Activate
    Wrd.Selection.Paste
endif
```

Работа с таблицами

Добавляем в четвертый абзац первого документа таблицу из трех строк и пяти колонок:

Wrd=createobject("Word.Application")
Wrd.documents.add
Wrd.visible=.t.
Wrd.Documents(1).tables.add (wrd.documents(1).paragraphs(4).range, 3,5)

Таблица является объектом-контейнером и состоит из колонок (columns) и строк (rows), пересечения колонок и строк дают ячейки (cell(i,j)).
Для того чтобы добавить строки или колонки пользуются методом add применительно к коллекциям rows или columns:
Команда
Wrd.documents(1).tables(1).rows.add добавляет строку.
Команда
Wrd.documents(1).tables(1).columns.add добавляет колонку.
Команда
? Wrd.documents(1).tables(1).columns.count
выдаст количество колонок в таблице.
Команда
Wrd.documents(1).tables(1).columns(2).width = 30
установит ширину второй колонки в 30 точек.
Высоту первой строки в 20 точек устанавливаем командой
wrd.documents(1).tables(1).rows(1).height = 20
Для ссылки на ячейки таблицы используют объект cell. Так, например, для того чтобы занести текст в третью ячейку первой строки, распечатать его целиком, а затем, распечатать его первое слово, можно воспользоваться командами
wrd.activedocument.tables(1).cell(1,3).range.text = "Привет, это пробный текст"
? wrd.activedocument.tables(1).cell(1,3).range.text
? wrd.activedocument.tables(1).cell(1,3).range.words(1).range.text

Форматирование таблицы целиком, строк, колонок, отдельных ячеек таблицы производится через свойства shading (заливка), borders (границы), format. Так, например, для того чтобы закрасить всю таблицу красным цветом можно применить следующие команду
wrd.activedocument.tables(1).Shading.BackgroundPatternColor = RGB(200,0,0)

Свойства шрифта задаем через свойство format объекта range
wrd.ActiveDocument.Tables(1).Range.Font.Size = 10
wrd.ActiveDocument.Tables(1).Range.Font.Color = RGB(0,255,0)

Границы можно устанавливать у таблицы целиком, рядов, строк или отдельных ячеек применяя свойство LineStyle объекта borders соответствующих объектов. LineStyle = 1 соответствует сплошной линии, LineStyle = 0 – линия отсутствует. Ширина линии определяется свойством LineWidth. При использовании объекта borders(WdBorderType) необходимо указать тип границы. Список стандартных переменных Word Basic соответствующих типам границы и их числовые значения приведены в следующей таблице

WdBorderType	Название константы Word Basic	Описание
-1	wdBorderTop	Границы сверху

-2	wdBorderLeft	Границы слева
-3	wdBorderBottom	Границы снизу таблицы
-4	wdBorderRight	Границы справа
-5	wdBorderHorizontal	Горизонтальные границы между ячейками (строками)
-6	wdBorderVertical	Вертикальные границы между ячейками (столбцами)
-7	wdBorderDiagonalDown	Диагональ вниз
-8	wdBorderDiagonalUp	Диагональ вверх

Данный фрагмент кода устанавливает все границы и зачеркивает по диагонали все ячейки таблицы

```
With wrd.activedocument.tables(1)
    .Borders( -1).LineStyle = 1
    .Borders( -2).LineStyle = 1
    .Borders( -3).LineStyle = 1
    .Borders( -4).LineStyle = 1
    .Borders( -5 ).LineStyle = 1
    .Borders( -6).LineStyle = 1
    .Borders( -7).LineStyle = 1
    .Borders( -8).LineStyle = 1
endwith
```

Работа с плавающими объектами. Изображения, рамки с текстом

```
? wrd.Documents(1).shapes.count
```

Встроенное в абзац изображение представляет собой объект класса InlineShape.
Количество внедренных изображений получаем командой:

```
? wrd.ActiveDocument.InlineShapes.Count
```

Если у изображения изменить режим обтекания с «в тексте» на любой другой режим: «вокруг рамки», «по контуру», «за текстом», «перед текстом», то тип объекта-изображения меняется, оно становится объектом shape.

```
? wrd.ActiveDocument.Shapes.Count
```

Информацию о типе плавающего объекта можно получить используя свойство name

```
? wrd.ActiveDocument.Shapes(1).name
```

Если в названии объекта есть слово "Picture" – это изображение, если – "Text Box" – рамка с текстом, если – "Rectangle" – рисованный прямоугольник и т.д.

Горизонтальное и вертикальное положение изображения определяется свойствами left и top. Способ вычисления координат определяется свойствами RelativeHorizontalPosition и RelativeVerticalPozition.
Например:
Команды

```
wrd.documents(1).shapes(1).relativehorizontalposition=1
wrd.documents(1).shapes(1).relativeverticalposition=1
```

устанавливает способ задания горизонтального положения относительно левого среза страницы документа, а вертикального – относительно верхнего среза листа.

Команды
wrd.documents(1).shapes(1).left=0
wrd.documents(1).shapes(1).top = 0
помещают изображение в левый верхний угол листа.
Цвет заливки и прозрачность заливки рисованного объекта можно задать следующим образом:
wrd.ActiveDocument.Shapes(1).Fill.ForeColor = 255
wrd.ActiveDocument.Shapes(1).Fill.Transparency = 0.5
цвет можно задавать макросом RGB(nR,nG,nB), прозрачность – числовой величиной в интервале от 0 до 1.

Тип и настройки изображения задаются свойствами объекта PictureFormat – ColorType, Brightness, Contrast.
wrd.ActiveDocument.Shapes(1).PictureFormat.ColorType = 3 – ч/б изображение
wrd.ActiveDocument.Shapes(1).PictureFormat.ColorType = 2 – серое изображение
wrd.ActiveDocument.Shapes(1).PictureFormat.Brightness = 0.5 – яркость 50%
wrd.ActiveDocument.Shapes(1).PictureFormat.Contrast = 0.5 – контрастность – 50%

У объектов shape можно менять порядок при помощи метода Zorder(ZOrderCmd). Таблица значений параметра ZOrderCmd приведена ниже:

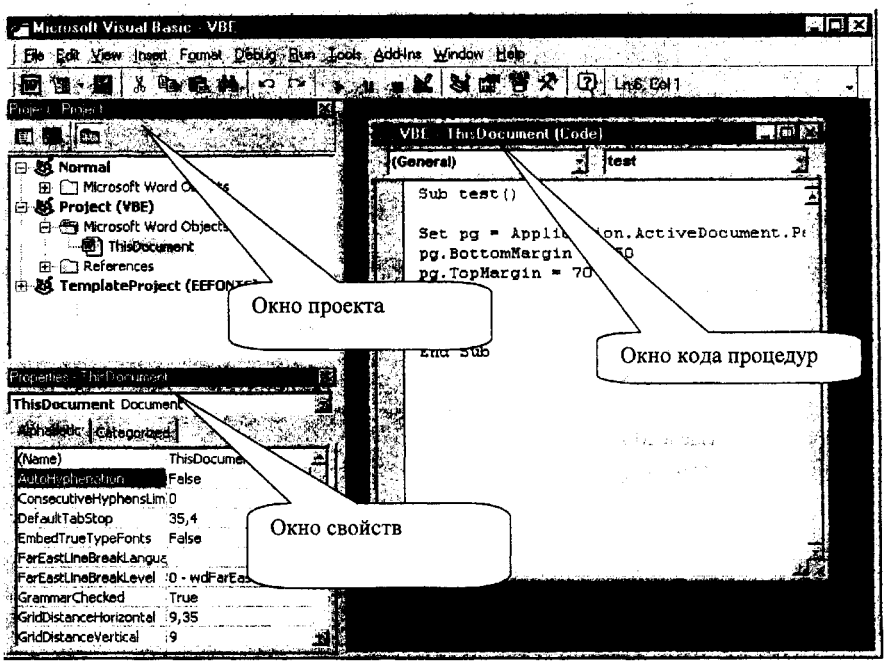
ZOrderCmd	Название константы ZOrderCmd	Описание действия
0	msoBringToFront	Переместить на передний план
1	msoSendToBack	Переместить на задний план
2	msoBringForward	Переместить вперед
3	msoSendBackward	Переместить назад
4	msoBringInFrontOfText	Расположить перед текстом
5	msoSendBehindText	Расположить после текста

Кроме изображений представляет интерес использование плавающих объектов – рамок с текстом – "Text Box". В текстовую рамку можно поместить (или считать из нее) некоторое текстовое значение (если есть необходимость использовать числа – то их предварительно нужно преобразовать в текст).
Если shapes(3) рамка с текстом, тогда следующие команды получают или задают его содержимое:
? wrd.ActiveDocument.Shapes(3).TextFrame.TextRange.text.text
получаем весь текст целиком
? wrd.ActiveDocument.Shapes(3).TextFrame.TextRange.words.count
определяем сколько слов в текстовой рамке
? wrd.ActiveDocument.Shapes(3).TextFrame.TextRange.words(1).text
распечатываем первое слово из текста.

Редактор VBA

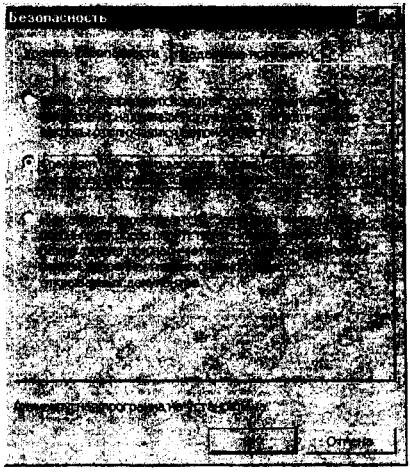
Для управления объектами MS'Word, как и другими объектами объектной модели MS'Office, предназначен специальный язык программирования – Visual Basic for Application. Редактор VBA вызывается из основного меню MS'Word: **сервис-> макрос -> редактор Visual Basic**. В основном состоянии в редакторе, как правило, открыты окна проекта (Project), свойств (Properties) и окно кода процедур(Code). В окне кода можно писать тексты процедур. В окне проекта указывают объект, в который внедряется программный код. Обычно процедуры внедряют в объект ThisDocument текущего проекта, в этом случае процедуры будут физически расположены внутри doc- файла и будут работать только для данного документа. Если процедура создается внутри объекта Normal – она внедряется в файл общего шаблона

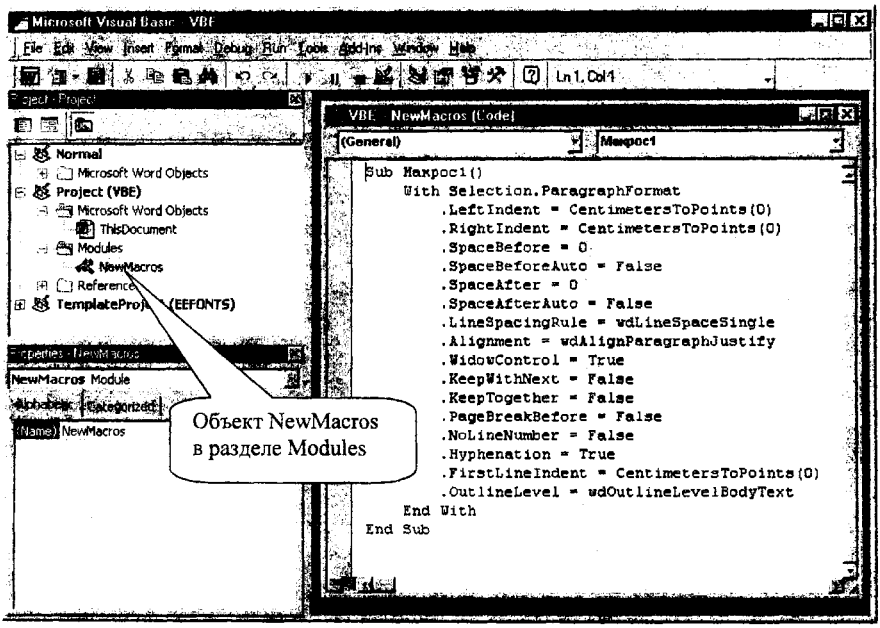
normal.dot и будет доступна для любого Word-документа открытого или созданного с этим шаблоном.

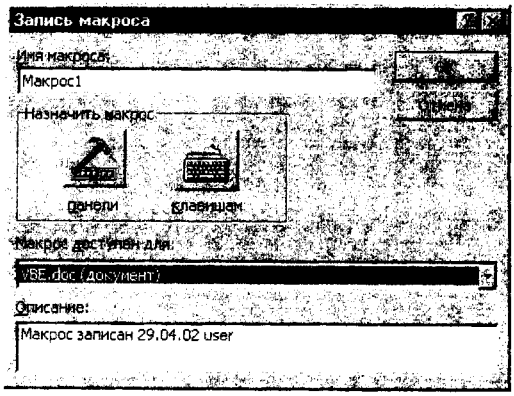


Встроенный редактор Visual Basic

В дальнейшем для работы с встроенным редактором VBA необходимо проверить установленный уровень безопасности в MS'Word. Рекомендуется установить средний уровень безопасности.







Если вы хотите узнать, как используется тот или иной объект или метод, вы можете воспользоваться средством для записи макросов – Macro Recorder. Включите Macro Recorder на запись и выполните те действия вручную. Macro recorder преобразует ваши действия в подпрограмму на Visual Basic. После того, как вы запишете ваши действия, вы можете открыть и модифицировать код макроса так, как вам нужно. Например, если вы не знаете, каким образом задаются отступы у параграфа, вы должны выполнить следующие действия:

1. В пункте **Tools** основного меню MS'Word, выберите пункт **Макрос (Macro)**, затем – **Начать запись... (Record New Macro)** .

2. Если вас не устраивает стандартное имя макроса – измените его и выберите место для сохранения макроса (либо файл шаблона – normal.dot, либо ваш документ – файл .doc) и начните запись нажатием клавиши **OK**.
3. Далее в меню **Формат (Format)** выберите **Абзац (Paragraph)**.
4. Измените величину левого отступа абзаца и нажмите **OK**.
5. Остановите запись макроса нажатием клавиши **Остановить запись (Stop Recording)** на панели управления записью.
6. В меню **Сервис (Tools)**, выберите пункт **Макрос (Macro)**, затем выберите пункт **Макросы (Macros)**.
7. Выберите имя макроса, которое вы дали ему в п.2 и нажмите кнопку **Изменить (Edit)**.

Посмотрите на код Visual Basic и найдите строки соответствующие изменению левого отступа абзаца (свойство **LeftIndent**).

Основы объектной модели Excel

Основными элементами объектной модели MS'Excel являются объекты и коллекции: Excel.Application, WorkBooks, WorkBooks(n), WorkSheets, WorkSheets(n), Cells(i,j)…

Создать объект MS'Excel можно также как и объект MS'Word – при помощи команды createobject:
Exl = createobject("Excel.application")

Объектная переменная exl представляет собой экземпляр основного окна MS'Excel, он обладает таким же набором свойств, как и основное окно MS'Word: visible, windowstate, top, left, width,height …Экземпляр MS'Excel включает в себя коллекцию рабочих книг – WorkBooks, которая первоначально пуста (свойство exl.workbooks.count имеет значение 0). Методом add можно добавить новую рабочую книгу, а методом open – открыть уже существующий файл.
Команда
? Exl.WorkBooks.Count
первоначально выдаст 0. После добавления рабочей книги командой
Exl.WorkBooks.Add
Количество рабочих книг в коллекции workbooks
? Exl.WorkBooks.Count
должно быть равным 1.
Если вы откроете существующий файл-рабочую книгу, например u:\test.exl,
Exl.workbooks.add("u:\tst.exl"), то количество рабочих книг будет равно двум.
Команды
? Exl.workbooks(1).name
возвращают имя рабочей книги.

Каждая рабочая книга, в свою очередь, обладает коллекцией рабочих листов. По умолчанию в MS'Excel версий 7-10 содержится три листа. Добавить лист в коллекцию можно методом – add:
Exl.workbooks(1).workshets.add
Получить и изменить имя листа можно при помощи свойства name:
? Exl.workbooks(1).workshets(1).name
exl.workbooks(1).workshets(2).name="Отчет"

Каждый лист содержит коллекцию ячеек – cells(nRow,nCol), здесь nRow, nCol – номер строки и колонки соответственно. Значение ячейки содержится в свойстве Value.
Так команды
Exl.WorkBooks(1).WorkSheets(1).Cells(1,1).Value="ФИО"
Exl.WorkBooks(1).WorkSheets(1).Cells(1,2).Value="Отр.дней"
Exl.WorkBooks(1).WorkSheets(1).Cells(1,3).Value="Оклад"
запишут в первую строку шапку таблицы: в ячейку A1 – «ФИО», B1 – «Отр.дней», C1 – «Оклад».
Для записи формул предназначены свойства Formula и FormulaR1C1. Следует заметить, что в VBA числа нужно записывать в американской нотации – разделитель десятичной части – точка, а не запятая даже для русифицированной версии Excel.
В свойство Formula задается выражение формулы, где ссылки записываются в стиле A1 (в виде буквы и цифры: A1, B12 и т.д.) В свойстве же FormulaR1C1 формулы записываются с использованием так называемой R1C1 нотации, в этой нотации нумеруются как строки, так и столбцы. В стиле ссылок R1C1 Microsoft Excel указывает положение ячейки буквой «R», за которой идет номер строки, и буквой «C», за которой идет номер столбца.
Exl.WorkBooks(1).WorkSheets(1).cells(2,4).Formula = "=B2*C2/24"
Exl.WorkBooks(1).WorkSheets(1).cells(2,4).Formula = "=0.15*B2*C2/24"

Exl.WorkBooks(1).WorkSheets(1).cells(2,4).Formula = "=R3C2*R3C3/24"
Exl.WorkBooks(1).WorkSheets(1).cells(2,4).Formula = "=0.15*R3C2*R3C3/24"

Основные элементы языка VBA

Типы данных

В VBA поддерживаются следующие типы данных:

Тип	**Описание**
Boolean	Логический тип данных. Могут принимать два значения: **true** (истинно) и **false** (ложно) true соответствует числовое значение -1, false 0
Byte	Байт. Тип целых данных в пределах от 0 до 255
Currency	Формат для денежных величин
DataObject	Объект для доступа к данным
Date	Формат для отображении даты и времени
Decimal	Десятичное число
Double	Вещественное число (число с плавающей точкой) двойной точности
Empty	Значение переменной по умолчанию. Задается при объявлении переменных до присваивания ему значения
Error	Формат для хранения номеров ошибок исполнения сценариев
Integer	Целое число (в диапазоне от - 32 768 до 32 767)
Long	Длинное целое число (в диапазоне от - 2 147 483 648 до 2 147 483 647)
Null	Переменная, не содержащая никаких данных
Object	Объект для доступа к данным OLE, ActiveX
Single	Вещественное число (число с плавающей точкой) обычной точности

String	Символьный тип данных

Переменные объявляются с помощью оператора **Dim**

Dim x,y,z

Константы объявляются с помощью оператора **Const**

Const pi = 3.14
Арифметические операторы
+, -, *, /, ^, MOD
Сложение строк (конкатенация) +, &
Логические операторы
And, Or, Not, Xor, Eqv, Imp

Для декларирования переменных объектного типа используется оператор set. Например, переменная X – ссылка на рабочий лист объявляется следующим образом.

Set X = CreateObject("Excel.Application")

Управляющие структуры

Безусловный цикл Each...Next

Цикл **For Each...Next** повторяет блок команд для каждого объекта из указанной коллекции или каждого элемента из массива. Следующий пример выводит в окно информацию о количестве символов в каждом открытом документе MS'Word:

```
Sub UUU()
Txt = ""
    For Each dc In Application.documents
        Txt = txt + dc.name +CStr(dc.characters.count) +"  символов"+chr(13)
    Next
MsgBox txt
End Sub
```

Цикл **For Each...Next** можно прервать при помощи команды **Exit For**. Эту команду обычно помещают внутрь оператора **If...Then...Else** или **Select Case.**

В следующем примере (для MS'Excel) проверяется диапазон ячеек A1:B5. Если в нем содержится ячейка с нечисловым значением, тогда выводится сообщение, а цикл прерывается.

```
Sub TestForNumbers()
Set aa = Application.Range("A1:B5")
ttt = 0
For Each u In aa
        If IsNumeric(u.Value) = False Then
            MsgBox "В ячейке содержится нечисловое значение"
            Exit For
        Else
            ttt = ttt + u.Value
        End If
Next
```

```
Txt = "Сумма значений ячеек диапазона = "+CStr(ttt)
MsgBox txt
End Sub
```

Безусловный цикл For... Next...

For [счетчик=начало] to [конец] Step [шаг]

[Команды тела цикла]

Next

Следующий фрагмент кода высчитывает сумму всех четных чисел от 2 до 100.

```
Sub EvenTotal()
    For j = 2 To 100 Step 2
        total = total + j
    Next j
    MsgBox "Сумма четных чисел от 2 до 100 равна " & total
End Sub
```

Шаг цикла может быть отрицательным.

```
Sub NewTotal()
    For myNum = 16 To 2 Step -2
        total = total + myNum
    Next myNum
    MsgBox "Сумма = " & total
End Sub
```

Условный цикл Do ... Loop

Существует несколько разновидностей условного цикла Do … Loop:

Do

...

Loop Until [условие]

Выполняет код хотя бы один раз, пока условие не будет истинным.

Do

...

Loop While [условие]

Выполняет код хотя бы один раз, пока условие справедливо.

Do Until [условие]

...

Loop

Повторяет выполнение кода, пока условие не станет справедливым.

Do While [условие]

...

Loop

Повторяет выполнение кода, пока условие справедливо.

While [условие]

[действия]

Wend

Выполняет код, пока условие истинно.

В нижеприведенной процедуре `ChkFirstWhile`, условие проверяется до входа в цикл. В результате будет выведено сообщение, что, цикл выполнился 0 раз, а величина myNum, естественно, равна 9. Во втором примере – процедуре `ChkLastWhile`, условие проверяется после выхода из цикла, поэтому, несмотря на то, что `myNum < 10`, цикл один раз все-таки выполнится.

```
Sub ChkFirstWhile()
    counter = 0
    myNum = 9
    Do While myNum > 10
        myNum = myNum - 1
        counter = counter + 1
    Loop
    MsgBox "Цикл выполнился" & counter & " раз. MyNum="&MyNum
End Sub

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

Аналогично тому как использовалась фраза While, условие **Until** также может проверяться как вначале, так и в конце цикла.

```
Sub ChkFirstUntil()
    counter = 0
    myNum = 20
    Do Until myNum = 10
        myNum = myNum - 1
        counter = counter + 1
    Loop
```

```
    MsgBox "Всего выполнено " & counter & " повторений."
End Sub

Sub ChkLastUntil()
    counter = 0
    myNum = 1
    Do
        myNum = myNum + 1
        counter = counter + 1
    Loop Until myNum = 10
    MsgBox "Всего выполнено " & counter & " повторений."
End Sub
```

Из цикла **Do...Loop** можно также выйти по условию, используя команду **Exit Do**. Например, используя команду **Exit Do,** помещенную в оператор **If...Then...Else** или **Select** можно выходить из бесконечного цикла и т.д.

```
Sub ExitExample()
    counter = 0
    myNum = 9
    Do Until myNum = 10
        myNum = myNum - 1
        counter = counter + 1
        If myNum < 10 Then Exit Do
    Loop
    MsgBox "Цикл сделал " & counter & " повторений."
End Sub
```

Цикл While ... Wend

While *condition*
[*statements*]

Wend

Цикл выполняется до тех пор, пока выражение цикла – *condition* возвращает **True**. Если *condition* равно **True**, будут выполняться операторы тела цикла. После того, как процесс дойдет до команды **Wend** управление передается в начало цикла – к оператору **While** и вновь проверяется условие. Если условие до сих пор верно, цикл вновь повторится. Если условие примет значение **False**, выполнение цикла останавливается и начинает выполняться команда следующая за оператором **Wend**.

Циклы **While...Wend** могут быть вложенными. Каждый оператор **Wend** обозначает завершение своего цикла **While**. Рекомендуется вместо цикла While ... Wend использовать цикл **Do...Loop**, который обладает большей гибкостью.

Оператор ветвления If...Then...Else

Условный оператор **If ... Then ... Else** используется для выполнения действий, в зависимости от нескольких условий.

If [условие] Then
код выполняется, если условие выполнено
Else

код выполняется, если условие не выполнено
End If
В качестве примера составим процедуру выводящую приветствие, в зависимости от времени суток:
Sub Hi_()
H = Hour(now) 'получаем текущий час
If H < 12 Then
  MsgBox "Доброе утро"
Else If H < 17 Then
  MsgBox "Добрый день"
Else
  MsgBox "Добрый вечер"
End If
End If
End sub

В VBA можно использовать сокращенную форму оператора If:

```
If [условие] Then [оператор]
Для одного оператора или

If [условие] Then
   операторы
End If
```

Множественное ветвление

В случае необходимости организации множественного ветвления, можно добавлять в оператор **If** дополнительные блоки при помощи фразы **ElseIf**. Например, в следующем примере функция вычисляет премию(bonus) в зависимости от категории работы(performance):

```
Function Bonus(performance, salary)
    If performance = 1 Then
        Bonus = salary * 0.1
    ElseIf performance = 2 Then
        Bonus = salary * 0.09
    ElseIf performance = 3 Then
        Bonus = salary * 0.07
    Else
        Bonus = 0
    End If
End Function
```

Для организации более гибкого множественного ветвления в VBA существует специальный оператор **Select Case.**

Select Case (выражение)
Case (значение1)
    (действие1)
Case (значение2)
    (действие2)
Case Else
(операторы иначе...)

End Select

В следующем примере при помощи оператора **Select Case** вычисляется премия в зависимости от категории работы:

```
Function Bonus(performance, salary)
    Select Case performance
        Case 1
            Bonus = salary * 0.1
        Case 2, 3
            Bonus = salary * 0.09
        Case 4 To 6
            Bonus = salary * 0.07
        Case Is > 8
            Bonus = 100
        Case Else
            Bonus = 0
    End Select
End Function
```

Операторные скобки With

В VBA для облегчения написания программного кода и улучшения читаемости также как и в VFP существуют операторные скобки **With:**

Процедура в следующем примере заполняет диапазон ячеек числом 30 и форматирует текст.

```
Sub FormatRange()
    With Worksheets("Sheet1").Range("A1:C10")
        .Value = 30
        .Font.Bold = True
        .Interior.Color = RGB(255, 255, 0)
    End With
End Sub
```

Можно использовать вложенные скобки **With** для большей эффективности:

```
Sub MyInput()
    With Workbooks("Книга1").Worksheets("Лист1").Cells(1, 1)
        .Formula = "=SQRT(50)"
        With .Font
            .Name = "Arial"
            .Bold = True
            .Size = 8
        End With
    End With
End Sub
```

Основные функции

Преобразование типов данных

Функции преобразования используются для перевода значений переменных из одних типов данных в другие

Функция	Значение
Abs(*число*)	Абсолютное значение *числа*
Asc(*символ*)	ANSI код *символа*
AscW(*символ*)	то же для *символов* Unicode
Chr(*число*)	Возвращает символ по коду
ChrW(*число*)	то же для символов Unicode
CBool()	Преобразование в логический тип
CByte()	Преобразование в число байт
CDate()	Преобразование в формат даты
CDbl()	Преобразование в вещественное число двойной точности
CInt()	Преобразование в целое число
CLng()	Преобразование в длинное целое
CSng()	Преобразование в вещественное число обычной точности
CStr()	Преобразование в строку символов
Fix(*число*)	Возвращает целую часть *числа*
Hex(*число*)	Преобразование *числа* в шестнадцатеричное счисление
Int(*число*)	Возвращает целую часть *числа*
Oct(*число*)	Преобразование *числа* в восьмеричное счисление
Round(*число*)	Округление числа до заданного *числа* десятичных позиций
Sgn(*число*)	Возвращает целое число, отражающее знак аргумента

Основные математические функции

Функция	Значение
Atn(*число*)	арктангенс *числа*
Cos(*число*)	косинус *числа*
Exp(*число*)	Экспонента *числа*
Log(*число*)	Натуральный логарифм *числа*
Randomize	Устанавливает генератор случайных чисел в исходное состояние
Rnd()	Случайное число от 0 до 1
Sin(*число*)	Синус *числа*
Sqr(*число*)	Квадратный корень *числа*
Tan(*число*)	Тангенс *числа*

Функции даты и времени

Данные функции служат для работы со значениями даты и времени на системных часах компьютера

Функция	**Описание**

Date	Текущая дата
DateAdd	Дата с добавленным заданным временным интервалом
DateDiff	Возвращает интервал между двумя заданными датами
DatePart	Возвращает только день, месяц или год заданной даты
DateSerial	Возвращает значение в формате Date для заданных года, месяца и дня
DateValue	возвращает значение в формате Date
Day	день месяца
Hour	Час дня
Minute	Минута
Month	Номер месяца
MonthName	Название месяца по его номеру
Now	Текущие дата и время
Second	Секунда
Time	Текущее время
TimeSerial	данные в формате Date для заданных часа, минуты и секунды
TimeValue	время в формате Date
WeekDay	номер дня недели (воскресенье = 1)
WeekDayName	Название дня недели по его номеру
Year	Год

Строковые функции

Функция	**Описание**
Filter	Возвращает массив, отобранный по заданному критерию из заданного массива строк
FormatCurrency	Преобразует строку для предоставления денежных сумм
FormatDateTime	Преобразует строку для предоставления даты и времени
FormatNumber	Форматирует строку как число
FormatPercent	Преобразует строку для предоставления процентного соотношения
InStr	Возвращает место первого появления одной строки внутри другой
InStrRev	то же, но начинается с конца строки
Lcase	Преобразовывает символы строки в строчные буквы
Left	Возвращает заданное количество символов от левого конца строки
Len	Возвращает длину строки
Ltrim	Удаляет начальные пробелы
Mid	Возвращает заданное количество символов из строки
Replace	Замена символов в строке
Right	Возвращает заданное количество символов от правого конца строки
Rtrim	Удаляет конечные пробелы
Space	Строка состоящая из заданного количества пробелов
String	Строка состоящая из заданного количества одного символа
StrReverse	"Отраженная" строка
Trim	Удаляет начальные и конечные пробелы
Ucase	Преобразует символы строки в прописные

Функции проверки переменных

Данные функции используются для определения типа данных, находящихся в переменной. Возвращают true или false.

Функция	**Описание**

IsArray	Массив
IsDate	Дата/время
IsEmpty	Инициализированная ли переменная
IsNull	Содержит ли некорректные данные
IsNumeric	Является ли числом
IsObject	Является ли объектом OLE или ActiveX

Служебные символы для работы со строками

Значение	**Описание**
Chr(10)	Перенос строки
Chr(13)	Возврат каретки
Chr(13)&Chr(10)	Возврат каретки и перенос строки
Chr(9)	Горизонтальная табуляция

Диалоговые окна

Функция **MsgBox** выводит окно сообщения и может возвращать значение, какая кнопка была нажата пользователем. Синтаксис:

MsgBox [текст сообщения], [параметры картинки и кнопок],[заголовок окна]

Параметры, определяющие картинку и кнопки окна.

Константа	**Значение**	**Описание**
VBOKOnly	0	Кнопка OK
VBOKCancel	1	Кнопки OK и Отмена (Cancel)
VBAbortRetryIgnore	2	Кнопки Стоп (Abort), Повтор(Retry), Пропустить(Ignore)
VBYesNoCancel	3	Кнопки Да (Yes), Нет(No), Отмена (Cancel)
VBYesNo	4	Кнопки Да (Yes), Нет(No)
VBRetryCancel	5	Кнопки Повтор(Retry), Отмена(Cancel)
VBCritical	16	Иконка важного сообщения
VBQuestion	32	Иконка вопросительного сообщения
VBExclamation	48	Иконка предупреждения
VBInformation	64	Иконка информационного сообщения
VBDefaultButton1	0	По умолчанию активна первая кнопка
VBDefaultButton2	256	По умолчанию активна вторая кнопка
VBDefaultButton3	512	По умолчанию активна третья кнопка
VBDefaultButton4	768	По умолчанию активна четвертая кнопка
VBApplicationModal	0	Пока пользователь не нажмет кнопку, работа приложения останавливается
VBSystemModal	4096	Вся система останавливается, пока пользователь не нажмет кнопку

Для задания сочетаний картинок и кнопок нужно сложить соответствующие константы или значения. Например, требуется спросить у пользователя, нужно ли завершить работу? Для этого нам понадобится иконка вопроса и кнопки Да-Нет.

MsgBox "Завершить работу?", vbYesNo+vbQuestion, "Вопрос"

или сложив числовые значения 4 + 32 = 36

MsgBox "Завершить работу?", 36, "Вопрос"

Мы можем определить, какая кнопка выбрана, присваивая переменной значение возвращаемое функцией **MsgBox**

Button_result = MsgBox("Завершить сценарий?", 36, "Вопрос")

Значения, возвращаемые функцией **MsgBox**

Константа	Значения	Нажатая кнопка
VBOK	1	OK
VBCancel	2	Отмена (Cancel)
VBAbort	3	Стор (Abort)
VBRetry	4	Повтор (Retry)
VBIgnore	5	Пропустить (Ignore)
VBYes	6	Да (Yes)
VBNo	7	Нет (No)

Окно **MsgBox** позволяет также подключить файл справки, для этого в конце конструкции добавляется путь к файлу справки:
MsgBox "Завершить сценарий?", 36, "Вопрос", "C:\Help.hlp", 123
InputBox(Ввод информации)
Служит для ввода пользователем данных и использования их сценарием. Введенные данные воспринимаются как строка (string).
Синтаксис:
InputBox([Message],[Title],[InputMess], x,y)
где
Message - сообщение в окне
Title - заголовок окна.
InputMess - сообщение в строке ввода. Если нужно, чтобы строка ввода была пустой ставятся двойные кавычки.
x и y - координаты положения окна на экране.

login = InputBox ("Введите свое имя", "Login" ,"",500, 1500)
Так же как и **MsgBox**, окно ввода **InputBox** позволяет подключить файл справки.

Массивы

Массив – тип данных предназначенный для хранения набора переменных одного и того же типа данных. В отличие от Visual FoxPro в VBA нельзя создать массив из величин разного типа данных. Для объявления массива используется команда dim
Команда `Dim curExpense(364) As Currency` объявляет массив из 365 элементов. Индекс массива может изменяться от 0 до 364.
Следующий пример инициализирует массив значениями 20.

```
Sub FillArray()
    Dim curExpense(364) As Currency
    Dim intI As Integer
    For intI = 0 to 364
        curExpense(intI) = 20
    Next
End Sub
```

Можно изменять нижний индекс массива так, что бы номер первого элемента был 1 при помощи команды Option Base, например команды:

```
Option Base 1
Dim curExpense(365) As Currency
```

объявляют массив curExpense из 365 элементов, причем номер первого элемента – 1.
Можно явно указать нижнюю границу индекса массива, используя фразу **to**.

```
Dim curExpense(1 To 365) As Currency
Dim strWeekday(7 To 13) As String
```

В VBA существует тип данных **Variant** для работы с неопределенными значениями. Массивы из переменных типа **variant** можно создать двумя способами. Один способ – объявить массив из переменных типа данных variant:

```
Dim varData(3) As Variant
```

```
varData(0) = "Петров Иван Сергеевич"
varData(1) = "ул. Ленина 145, 35"
varData(2) = 23
```
Второй способ состоит в том, чтобы сначала создать переменную типа variant, а затем при помощи функции **Array** задать значения массива:
```
Dim varData As Variant
varData = Array("Петров Иван Сергеевич", "ул. Ленина 145, 35", 38)
```
В Visual Basic, можно декларировать многомерные массивы размерностью до 60. Например, следующий пример декларирует 2-мерный массив 5 на 10.
```
Dim sngMulti(1 To 5, 1 To 10) As Single
```
Следующий пример заполняет двумерный массив, используя два вложенных цикла
```
Sub FillArrayMulti()
    Dim intI As Integer, intJ As Integer
    Dim sngMulti(1 To 5, 1 To 10) As Single

    For intI = 1 To 5
        For intJ = 1 To 10
            sngMulti(intI, intJ) = intI * intJ
            Debug.Print sngMulti(intI, intJ)
        Next intJ
    Next intI
End Sub
```

Основы Windows Script Host

Windows Script Host – представляет собой универсальный сервер скриптовых приложений или сервер сценариев. В настоящее время, начиная с Windows 98, WSH является составной частью операционной системы, физически WSH представляет собой файлы CSript.exe и WScript.exe которые расположены в системных директориях.
Установленный Windows Script Host поддерживает несколько видов файлов: vbs, vbe, js, jse, wsf, wsc и wsh. Все они (кроме vbe и jse) являются простыми текстовыми файлами и могут редактироваться в любом текстовом редакторе.
Файлы .vbs и .js являются файлами написанными на языке сценариев MS Visual Basic Script и MS JScript соответственно.
vbe и jse-файлы – это vbs и js-файлы зашифрованные с помощью программы MS Script Encoder
Файлы с расширением .wsf – это файлы содержащие XML-разметку для работы с WSH.
Файлы wsc - Windows Script Components (WSC) позволяют упаковывать сценарии для использования их в качестве COM-компонентов. По сути, это те же wsf-файлы, еще и содержащие COM-компоненты.
wsh–файлы являются файлами настроек Сервера Сценариев и выполняют по сути ту же роль, что и .pif файлы для DOS программ.
Как и .pif-файлы содержат настройки приложений DOS, так и .wsh-файлы содержат настройки для всех вышеперечисленных файлов. Обладая только несколько скромными возможностями по сравнению с PIF.
Для иллюстрации создадим пустой файл с любым из вышеперечисленных расширений (vbs, js или wsf).
Пусть это будет Example1.vbs. Щелкнем на нем правой кнопкой мыши и в окне свойств файла выберем вкладку “Сценарий” и увидим окно:
Изменим настройки на этой вкладке, поставив или сняв любой флажок, чтобы кнопка “Вернуть установки по умолчанию” стала доступна. После этого щелкним OK.
У нас появился файл с расширением wsh – Example1.wsh.
В дальнейшем, если мы хотим использовать измененные нами настройки, нужно вместо файла Example1.vbs нужно запускать файл Example1.wsh.

Откроем файл Example.wsh с помощью Блокнота и увидим следующий текст:
[ScriptFile]
Path=C:\Example1.vbs
Timeout=5
DisplayLogo=1
BatchMode=0
Параметр Path в разделе [ScriptFile] содержит путь к файлу, для которого искользуестя wsh-файл.
Настройки в разделе [Options] – это сами настройки, ради которых, собственно и создан файл настройки.
Timeout – определяет время отведенное для выполнения сценария.
DisplayLogo – отвечает за вывод эмблемы Windows Script Host, при запуске в командном режиме. Если изменить его на 0, то эмблема отображаться не будет.
BatchMode – включает/выключает пакетный режим. Если ему присвоить значение 1, то сценарий будет выполнятся в пакетном режиме – без вывода информации на экран и сообщений об ошибках.

Запуск сценариев (WScript.exe и CScript.exe)

Для запуска сценариев, в составе Windows Scripting Host служат файлы WScript.exe (диалоговый режим) и CScript.exe (режим командной строки).
WScript.exe служит для запуска сценариев из Windows. Используя его, вы можете запускать сценарии подобно обычным приложениям Windows. Вот несколько способов:

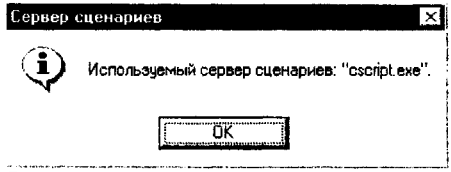
1. Запускать сценарий, как обычное приложение двойными щелчком мыши, выделить и нажать Enter и т.д.
2. Ввести имя файла сценария и путь к нему в окне в окне "Выполнить" (RUN) меню "Пуск" (Start).
3. Ввести в строку окна "Выполнить" WScript.exe и имя файла сценария (с указанием пути к нему). При этом вы можете использовать параметры запуска WScript.exe (см. ниже).
4. CScript.exe это версия Windows Scripting Host, которая используется для запуска сценариев из командной строки.

Синтаксис:
CScript [параметры] имя_файла.расширение [аргументы]
Для запуска сценариев, как с помощью CScript.exe так и с WScript.exe, можно использовать следующие параметры командной строки:

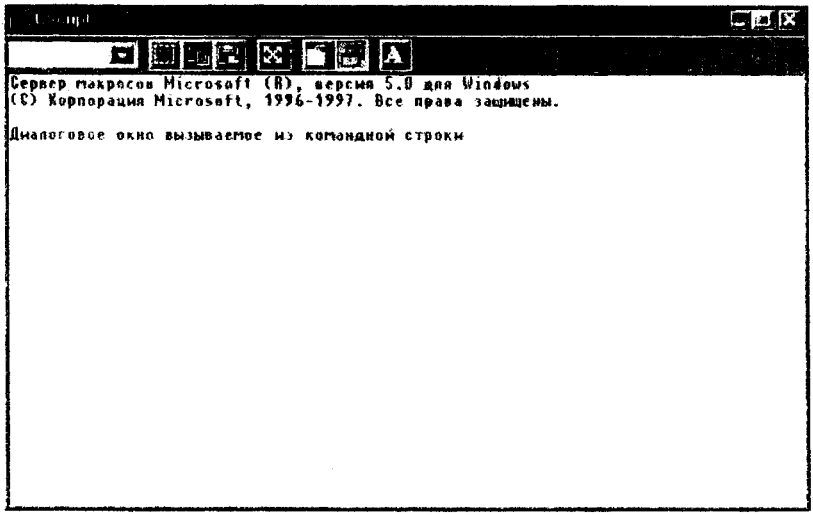
Параметр	Версия WSH	Описание
//B	1.0	Пакетный режим (подавляется вывод информации, запросов и сообщений об ошибках)
//D	2.0	Включить активную отладку
//E:*язык*	2.0	Указать язык сценария для исполнения файла
//H:CScript	1.0	Заменить исполняемый сервер сценариев на CScript.exe
//H:WScript	1.0	Заменить исполняемый сервер сценариев на WScript.exe
//I	1.0	Диалоговый режим (противоположный //B) (по умолчанию)
//Job:xxxx	2.0	Выполнить задание xxxx WSF-файла
//Logo	1.0	Отображать заставку (по умолчанию)
//Nologo	1.0	Не выводить заставку
//S	1.0	Запомнить параметры текущей командной строки для данного пользователя

//T:nn	1.0	Задать время исполнения сценария в секундах
//X	2.0	Выполнить сценарий под отладчиком
//U	2.0	Применять кодировку Unicode для перенаправленного консольного ввода-вывода

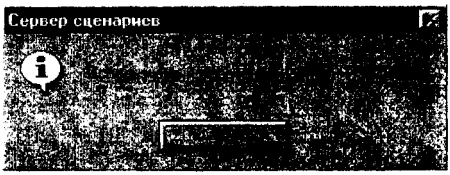
Например:
Cscript //T:10 MyScript.vbs
Отвести на выполнение сценария MyScript.vbs 10 секунд времени.
Попробуем сделать используемым сервером сценариев по умолчанию CScript.exe.
Для этого введем в командной строке:
wscript //h:cscript
Получим на экране:



С этого момента, все сценарии будут запускаться в режиме командной строки. Для многих сценариев выводящих информацию это будет несколько неудобно.
Попробуем запустить сценарий VBScript выводящий окно сообщения, состоящий из следующей инструкции:
WScript.Echo "Диалоговое окно вызываемое из командной строки"
Щелкнув мышью на файле сценария, получим:



Прочитать надпись практически невозможно, поскольку окно быстро "промелькнет" на экране.
Чтобы снова сделать сервером по умолчанию WScript.exe, введем в командную строку:
wscript //h:wscript
Появится окно:



И снова запускаемые файлы сценариев по умолчанию запускаются в диалоговом режиме. Нельзя сказать какой способ лучше нельзя сказать однозначно. У каждого есть свои плюсы и минусы.
В пользу WScript.exe говорит его интерактивность. В пользу CScript.exe – незаметность для пользователя. Хотя последнее спорно. Для незаметности лучше всего включить пакетный режим параметром //b.
Введем в командной строке:
wscript //b //s
Получим на экране окно:


Этим вы включим пакетный режим и сохраним настройки. Отныне все запущенные сценарии будут незаметны для пользователя – все сообщения об ошибках будут подавляться. Следует только позаботится, чтобы в сценарии не было ошибок иначе они не будут работать.
Но даже при включенном пакетном режиме у сценариев есть возможность выводить на экран управляющие окна.
Для отмены пакетного режима и сохранения настроек нужно включить интерактивный режим параметром //i и сохранить настройки:
wscript //i //s

Объекты Windows Scripting Host

В работе WSH используются 9 объектов: WScript (не путать с WScript.exe!), WshArguments, WshEnvironment, WshNetwork, WshShell, WshShortcut, WshSpecialFolders, WshUrlShortcut и FileSystemObject.
Объект WScript является главным объектом Windows Script Host. Он служит для создания объектов и выполняет служебные задачи связанные с ними, содержит сведения о сервере сценариев и о запущенных сценариях.
Объект WshArguments служит для работы с аргументами окружения
WshEnvironment – работает переменными окружения.
WshNetwork – используется при работе с сетевым окружением: содержит информацию для сети о данном компьютере, позволяет подключать сетевые принтеры и диски.
WshShell – служит для работы с переменными окружения Windows, запускает другие программы, работает с реестром и т.д.
WshShortcut – создает ярлыки.
WshSpecialFolders – используется для доступа к специальным папкам Windows, таким как меню Пуск, Рабочий стол, Мои документы и т.д.
WshUrlShortcut – еще один объект для создания ссылок, но обладающий более ограниченными возможностями, чем WshShortcut.
Особняком стоит FileSystemObject объект. Как таковой он не является объектом WSH и дочерним объектом WScript, но занимает важное место в создании сценариев используясь для работы с файлами.

Объект TextStream используется для работы с содержанием текстовых файлов.
Из всех вышеперечисленных объектов можно выделить 4 главных:WScript, WshShell, WshNetwork и FileSystemObject. Остальные же являются их объектами, созданными для удобства, дублируя некоторые их возможности.
Перед использованием всех объектов (кроме WScript) нужно создать их экземпляр. Для этого используется метод CreateObject, объекта WScript.
Например, объект WshShell создается следующим образом:
Set WshShell = WScript.CreateObject("WScript.Shell")
Объект Word создаем командой
Set www = WScript.CreateObject("Word.Application")

Объект WScript

Объект **WScipt** содержит информацию о сервере сценариев и о самих, исполняемых файлах сценариев.
Name – выводит надпись: "Сервер сценариев"
Пример:
WScript.Echo WScript.Name
FullName – возвращает используемый сервер сцениериев (CScript.exe или WScript.exe) и полный путь к нему.
Результат будет типа:
C:\WINDOWS\WSCRIPT.EXE
Path – возвращает путь к папке с файлами сервера сценариев (CScript.exe и WScript.exe). Если Windows находится в папке Windows, то результат будет:
C:\WINDOWS
Version – показывает версию установленного сервера сценариев. Обратите внимание, что свойство Version возвращает не версию языка Windows Script Host, а версию его интерпретатора.
Например
WScript.Echo WScript.Version
Выдаст результат: (5.0 или, 5.1), где 5.0 соответствует версии Windows Scriptinh Host 1.0, а 5.1. версии 2.0.
ScriptName – выдает имя исполняемого файла сценария.
WScript.Echo "Имя запущенного сценария: " + WScrlpt.ScriptName
ScriptFullName – возвращает полный путь и имя исполняемого файла сценария.
WScript.Echo "Путь к запущенному сценарию: " + WScript.ScriptFullName
Timeout
Свойство **Timeout** устанавливает время, по истечении которого сценарий завершает свою работу.
Данное свойство можно применять в тексте сценария только в версии 2.0, в версии 1.0 оно применяется только в WSH-файлах.
Синтаксис:
WScript.Timeout = time
где
time - время, отведенное на работу сценария в секундах.
устанавливаем время работы сценария:
WScript.Timeout = 5
WScript.Echo "Сценарий завершит работу через 5 секунд"
Interactive
Свойство **Interactive** показывает, используется ли диалоговый режим (WScript.exe), возвращая логический результат. А также может устанавливать или отключать диалоговый режим. При значении false - интерактивный режим отключается, т.е. диалоговые окна не могут использоваться.

Синтаксис:
WScript.Interactive[= True|False]
```
WScript.Echo("Диалоговый режим включен")  'отключаем диалоговый режим:
WScript.Interactive = false
WScript.Echo("Диалоговый режим отключен")  'данное окно не появится на экране
'включаем диалоговый режим:
WScript.Interactive = true
WScript.Echo("Это снова диалоговый режим")
```

Методы объекта WScript

Echo – выводит диалоговое окно с сообщением пользователю. При использовании CScript.exe выводит строку с текстом. В качестве аргументов допускается вывод списка значений через запятую. Можно внедрять в выводимую строку спецсимволы например – chr(13) – перевод строки.

```
X=2500.25
WScript.Echo "Текст сообщения",chr(13)," Выводим число x=",x
```

CreateObject – создает экземпляр объекта ActiveX.
Синтаксис:
object.CreateObject(strProgID[,strPrefix])
где **object** – объект WScript., **StrProgID** – класс к которому принадлежит объект
Например, создадим объекты Word, Excel, Visual FoxPro

```
WScript.Echo "Создадим объект Word.Application"
Set Wrd = WScript.CreateObject("Word.Application")
Wrd.documents.add
Wrd.visible = true
WScript.Echo "Создадим объект Excel.Application"
Set Exl = WScript.CreateObject("Excel.Application")
Exl.WorkBooks.add
Exl.visible = true
WScript.Echo "Создадим объект VisualFoxpro.Application"
Set Vis_Fox  = WScript.CreateObject("VisualFoxpro.Application")
Vis_Fox.visible = true
WScript.Echo "Готово"
```

Sleep – переводит сценарий в неактивное состояние, на заданное время (в миллисекундах), после чего продолжает его работу.

```
WScript.Echo "Начать отсчет  периода в 2 сек "
WScript.Sleep 2000
WScript.Echo "2 секунды прошло "
```

Quit – завершает работу сценария. Необязателен.
При использовании VBScript можно использовать функции Visual Basic например, для вывода информации – MsgBox, а для ввода – InputBox

```
x=InputBox("Введите x")
```

```
if x>100 then
MsgBox("Число больше 100")
WScript.Quit
else
MsgBox CStr(x),32+4
' В данном случае MsgBox вызывается как процедура, поэтому скобки у аргументов не нужны
end if
WScript.Echo "Продолжение работы сценария"
```

Еще один пример на использование MsgBox, в данном случае MsgBox возвращает значение нажатой клавиши

```
x=InputBox("Введите x")
tt = "x="+CStr(x)
otv = MsgBox(tt,32+4)  'Здесь MsgBox вызывается как функция и скобки у аргументов нужны
rpl = " Нажата кнопка со значением = " + CStr(otv)
MsgBox(rpl)
```

Объект Wscript.Shell

Иногда для вывода используется управляющее окно **Popup**. Оно имеет те же возможности вывода информации, что и окно созданное с помощью метода **Echo**, но вдобавок ко всему обладает дополнительными возможностями. Метод **Popup**, является методом объекта **Wscript.Shell**

Синтаксис:

intButton = object.Popup(strText,[WaitSec],[strTitle],[natType]) где

object - объект Wscript.Shell

strText - само сообщение в данном окне

WaitSec - время (в секундах), по истечении которого окно закроется

strTitle - заголовок окна. Если отсутствует, то заголовок окна будет по умолчанию "Сервер сценариев".

natType - параметр определяющий картинку и кнопку в данном окне.

Параметры определяющие кнопку:

Значение	**Кнопки**
0	**OK**
1	**OK и Отмена(Cancel)**
2	**Стоп(Abort), Повтор(Retry), и Пропустить(Ignore)**
3	**Да(Yes), Нет(No), и Отмена(Cancel)**
4	**Да(Yes) и Нет(No)**
5	**Повтор(Retry) и Отмена(Cancel)**

Параметры определяющие рисунок:

Значение	**Рисунок**
16	Важное сообщение
32	Вопрос
48	Предупреждение
64	Информация

При закрытии окно popup возвращает значение intButton – которое содержит информацию о том какая кнопка была нажата.

Значения выбранных кнопок:

Значения	**Нажатая кнопка**
1	OK

2	Отмена (Cancel)
3	Стор (Abort)
4	Повтор (Retry)
5	Пропустить (Ignore)
6	Да (Yes)
7	Нет (No)

Интересной особенностью окна popup является его возможность закрываться по истечении заданного времени.

```
Dim Interval, WshShell
'устанавливаем время через которое окно закроется:
Interval = 5
'создаем объект WshShell для метода Popup:
Set WshShell = CreateObject("WScript.Shell")
WshShell.Popup "Это окно закроется через 5 секунд",Interval, "Окно Popup", 48
```

В результате исполнения сценария на экране появится диалоговое окно, которое закроется по истечении 5 секунд.

Ввод-вывод в режиме командной строки

В Windows Scripting Host 2.0 появились методы, позволяющие работать с вводом-выводом информации в сценариях, запущенных под CScript.exe. Все они принадлежат объекту WScript.

Метод **StdIn** – позволяет вводить данные в сценарий в режиме командной строки

Синтаксис:

WScript.StdIn

Метод **StdOut** – выводит данные в строку

Синтаксис:

WScript.StdOut

Метод **StdErr** – предназначен для вывода данных об ошибках

Синтаксис:

WScript.StdErr

Важно помнить, что все эти методы могут работать, только будучи запущены, под CScript.exe. Если попытаться запустить их в диалоговом режиме, то будет получено сообщение "Неверный дескриптор" (Invalid Handle).

```
WScript.StdOut.WriteLine("Введите что-нибудь и нажмите Enter")
x = WScript.StdIn.ReadLine
 If x = "" Then
WScript.Echo "Ничего не введено"
Else
WScript.Echo "Вы ввели " + x
End If
```

Метод **StdErr** работает подобно методу **StdOut**.

```
WScript.StdErr.WriteLine("Сообщение об ошибке")
```

При использовании метода **StdIn**, могут некорректно отображаться введенные в сценарий данные, если используется русская раскладка клавиатуры

Специальные папки Windows. Свойство SpecialFolders объекта Wscript.Shell

Для получения пути к специальным папкам Windows типа Мои документы, Рабочий стол и т.д. используется свойство **SpecialFolders** объекта **WshShell**.
Синтаксис:
object.SpecialFolders(objWshSpecialFolders)
где
object - объект WshShell
objWshSpecialFolders – специальная папка

objWshSpecialFolders	**Название папки**
Desktop	Рабочий стол
Favorites	Избранное
Fonts	Шрифты
MyDocuments	Мои документы
NetHood	Сетевое окружение
PrintHood	Принтеры
Programs	Программы, меню "Пуск"
Recent	Раздел просмотренных документов, меню "Пуск"
SendTo	Отправить
StartMenu	Меню "Пуск"
Startup	Автозагрузка
Templates	Шаблоны

Кроме вышеперечисленных папок в Windows 2000 доступны: AllUsersDesktop, AllUsersStartMenu, AllUsersPrograms, и AllUsersStartup.

В качестве примера составим сценарий, узнающий адрес папки "Рабочего стола".

```
Dim WshShell, DesktopPath
Set WshShell = CreateObject("WScript.Shell")
'получаем путь к рабочему столу:
DesktopPath = WshShell.SpecialFolders("Desktop")
MsgBox "Адрес рабочего стола: " + DesktopPat
```

Сетевое окружение. Объект Wscript.Network

Объект **Wscript.Network** используется, как явствует из его названия, для работы с сетью, установки конфигурации сетевого окружения - а именно для управления сетевыми дисками и принтерами.
Через объект **Wscript.Network** можно получить информацию о локальном компьютере, подключаться к дискам и принтерам в сети, устанавливать принтер по умолчанию и отключаться от сетевых дисков и принтеров.
Для его использования объект нужно создать.
Set WshNetwork = CreateObject("WScript.Network")
Для получения информации о компьютере, можно получить его сетевые атрибуты: имя пользователя, имя компьютера и его домен.
Свойства **Wscript.Network**:

Свойство	Описание
ComputerName	Имя компьютера
UserDamain	Домен
UserName	Имя пользователя

Для демонстрации этих свойств составим сценарий.

```
Dim WshNetwork, NetInfo
Set WshNetwork = CreateObject("WScript.Network")
NetInfo = "Имя пользователя = " & WshNetwork.UserName & Chr(10)
NetInfo = NetInfo+ "Имя компьютера = "& WshNetwork.ComputerName &Chr(10)
NetInfo = NetInfo + "Домен = " & WshNetwork.UserDomain
MsgBox  NetInfo
```

Сетевые диски

Для работы с сетевыми дисками Windows Script Host предоставляет несколько методов, которые позволяют получать сведения о подключенных сетевых дисках, подключать сетевые диски и отключаться от них.

EnumNetworkDrives - возвращает список подключенных сетевых дисков.

Синтаксис:

objDrives = object.EnumNetworkDrive

где *object* - объект Wscript.Network

objDrives - переменная, которой присваивается ссылка на коллекцию сетевых дисков

```
Dim WshNetwork, Drives
Set WshNetwork = WScript.CreateObject("WScript.Network")
Set Drives = WshNetwork.EnumNetworkDrives
WScript.Echo "Подключены сетевые диски: "
For i = 0 to Drives.Count - 1 Step 2
WScript.Echo "Диски " & Drives.Item(i) & " = " & Drives.Item(i+1)
Next
```

MapNetworkDrive - назначает заданную букву сетевому диску.

Синтаксис:

WshNetwork.MapNetworkDrive strLocalName, strRemoteName, [bUpdateProfile], [strUser], [strPassword]

где

strLocalName - назначаемая буква сетевого диска

strRemoteName - удаленное имя

Необязательные параметры:

bUpdateProfile - логическая величина определяющая сохранять ли сделанную настройку в пользовательской конфигурации.

strUser, strPassword - вы можете указать имя и пароль для доступа к диску.

```
'создаем объект WshNetwork:
Set WshNetwork = CreateObject("WScript.Network")
'назначаем сетевой ресурс Server\PublicFiles как сетевой диск Z:
WshNetwork.MapNetworkDrive "Z:", "\\Server\PublicFiles"
```

RemoveNetworkDrive - отключает сетевой диск

Синтаксис:

WshNetwork.RemoveNetworkDrive strName, [bForce], [bUpdateProfile]

где:

strName - имя диска

Необязательные параметры:

bForce - логический параметр. Если принимает значение true, то сетевой диск отключается, даже если он в настоящий момент используется данным компьютером.

bUpdateProfile - логический параметр, указывающий, сохранить ли сделанную настройку в пользовательском профиле.

Пример:

```
'создаем объект WshNetwork:
Set WshNetwork = WScript.CreateObject("WScript.Network")
'подключаем сетевой диск Z:
WshNetwork.MapNetworkDrive "Z:", "\\Server\PublicFiles"
```

'отключаем сетевой диск Z:
WshNetwork.RemoveNetworkDrive "Z:"

Работа с файлами. FileSystemObject

Для файловых операций применяется объект FileSystemObject. Он не является прямым дочерним объектом WSH, а предназначен для использования совместно с языками сценариев. Вообще сами по себе языки сценариев VBS и JScript не могут работать с файлами и производить какие-либо действия над ними. Работать с файлами могут элементы ActiveX. Но и VBScript и JScript могут создавать объекты ActiveX (т.н. контейнеры объектов) и работать с ними. Для этого и используется **FileSystemObject**.
Для создания элемента ActiveX служит метод **CreateObject**. Создадим для работы экземпляр объекта **FileSystemObject**.
Set Fso = CreateObject("Scripting.FileSystemObject")

Объекты и коллекции FileSystemObject

FileSystemObject содержит следующие объекты и коллекции:

Объект/Коллекция	Описание
FileSystemObject	Основной объект. Содержит методы и свойства, которые позволяют создавать, удалять, получать информацию, и управлять дисками, папками и файлами. Многие методы связанные с этим объектом дублируются в других объектах FSO, которые предусматриваются для удобства.
Drive	Объект. Содержит методы и свойства, которые позволяют собирать информацию о накопителях, имеющихся в системе, как например, имя диска и сколько свободного места на диске. Имейте в виду, что "диск" не обязательно является жестким диском, но может быть накопителем CD-ROM, виртуальным диском RAM, и так далее. Накопитель не должен обязательно присутствовать на данном компьютере, он может быть доступен и через сеть.
Drives	Коллекция. Включает в себя все диски на данном компьютере независимо от их типа (HDD, CD-ROM и т.д.).
File	Объект. Содержит методы и свойства, которые позволяют создавать, удалять, или перемещать файл. Также получают сведения об имени файла, пути к нему, и другие свойства.
Files	Коллекция. Содержит список всех файлов, находящихся в данной папке.
Folder	Объект. Содержит методы и свойства, которые позволяют создавать, удалять, или перемещать папки. Также получают сведения об имени папки, пути к ней, и другие свойства.
Folders	Коллекция. Содержит список всех папок, находящихся в конкретной папке.
TextStream	Объект. Позволяет читать и делать записи в текстовые файлы.

Работа с файлами

FileSystemObject (далее FSO) предоставляет некоторые возможности для работы с текстовыми файлами. Можно создавать, удалять, копировать и перемещать файлы.
Для создания текстового файла применяют метод **CreateTextFile**. Синтаксис:
FSO.CreateTextFile([, overwrite[, unicode]])
где:

filename – имя создаваемого файла.
Необязательные параметры:
owerwrite – логический параметр (true или false), указывает, перезаписывать ли уже существующий файл с таким именем.
unicode – логический параметр (true или false), определяющий кодировку создаваемого файла. По умолчанию используется кодировка ASCII. Если принимает значение true – то файл создается в кодировке unicode.

```
Dim FSO, MyFile
Set FSO = CreateObject("Scripting.FileSystemObject")
Set MyFile = fso.CreateTextFile("c:\testfile.txt", true)
```

Примечание. При использовании JScript, при указании пути, вместо одной косой черты ("\") следует ставить две ("\\"). Это правило следует соблюдать, потому что интерпретатор, когда встречает в тексте сценария одну косую черту, ожидает что за ней последует спецсимвол JScript, (см.справочник JScript) и выдает сообщение об ошибке.
Для проверки существования файлов служит метод FileExists.

```
Set FSO = CreateObject("Scripting.FileSystemObject")
If (FSO.FileExists("c:\autoexec.bat")) Then
MsgBox "Файл autoexec.bat существует"
Else
MsgBox "Файл autoexec.bat  не существует"
End If
```

Для других операций над файлом, таких как копирование, перемещение и удаление файл должен быть сначала "получен" методом **GetFile**.

```
Set FSO = CreateObject("Scripting.FileSystemObject")
Set file1 = FSO.GetFile("MyFile.txt")
```

Для копирования, перемещения и удаления файлов объект FSO предоставляет несколько методов, в зависимости от того какой объект будет использоваться для операций FSO или file. Они показаны в следующей таблице.

Действие	**Объект.Метод**
Перемещение	File.Move или FileSystemObject.MoveFile
Копирование	File.Copy или FileSystemObject.CopyFile
Удаление	File.Delete или FileSystemObject.DeleteFile

Копирование файлов.

```
Set FSO = CreateObject("Scripting.FileSystemObject")
Set file1 = FSO.CreateTextFile("c:\test.txt")
Set file2  = FSO.GetFile("c:\test.txt")
file2.Copy("c:\0\test1.txt")
```

Перемещение файлов осуществляется аналогично методу копирования.
Удаление файлов:

```
Set FSO = CreateObject("Scripting.FileSystemObject")
Set file1 = FSO.CreateTextFile("test.txt")
'закрываем файл (см.работа с содержанием файла)
'если не закрыть файл, то он не может быть удален методом Delete
file1.Close
MsgBox "Файл создан"
Set file2  = FSO.GetFile("test.txt")
file2.Delete
MsgBox "Файл удален"
```

Свойства файла

Синтаксис:
object.Size
где
object - File объект
С помощью объекта **FSO** нам доступны такие свойства файла, как его размер, время создания и т.д.
Size – возвращает размер файла в байтах

```
Set FSO = CreateObject("Scripting.FileSystemObject")
'получаем текущий файл:
Set file1 = FSO.GetFile(WScript.ScriptFullName)
'получаем размер:
fsize = file1.Size
MsgBox "Размер файла " & WScript.ScriptName & " : " & fsize &" килобайт"
```

DateCreated – время создания файла
DateLastAccessed - время последнего обращения
DateLastModified - время последнего изменения

```
Set FSO = CreateObject("Scripting.FileSystemObject")
'получаем текущий файл:
Set file1 = FSO.GetFile(WScript.ScriptFullName)
'получаем время создания:
DC = file1.DateCreated
info = "Файл " & WScript.ScriptName & " :" & Chr(10)
info = info & "Создан: " & DC & Chr(10)
'получаем время последнего открытия:
DLA = file1.DateLastAccessed
info = info & "Открыт: " & DLA  & Chr(10)
'получаем время последнего изменения:
DLM = file1.DateLastModified
info = info & "Изменен: " & DLM
MsgBox info
```

Работа с содержимым файла

Для работы с содержимым фала используется объект **TextStream** и его методы.
Работа по изменению содержимого текстового файла состоит из нескольких действий:
1. Открытие файла
2. Работа с содержимым
3. Закрытие файла.
1. Открытие файла может осуществляться несколькими методами **FSO - OpenTextFile** и **OpenAsTextStream**.
Оба данных метода работают одинаково, только при применении метода **OpenAsTextStream** требуется, чтобы текстовый файл уже существовал, в то время как **OpenTextFile**, если открываемый файл не существует он может создать его.
Рассмотрим их.
OpenTextFile
Синтаксис:
FSO.OpenTextFile(filename[, iomode[, create[, format]]])
где
filename – имя открываемого файла
Необязательные параметры:
iomode – режим открытия файла.
Принимает значения:

1 - файл открывается для чтения. Записывать в него нельзя.
2 - файл открывается для записи.
8 - файл открывается для добавления данных
create – логическая величина, определяющая, будет ли создан новый файл, если файла с указанным именем не существует. true – файл создается.
format – кодировка для открытия файла.
Принимает значения:
- 2 – открывается, используя системную кодировку
- 1 – открывается в кодировке Unicode
0 – открывается в кодировке ASCII.
Пример:
Set fso = CreateObject("Scripting.FileSystemObject")
Set f = fso.OpenTextFile("c:\testfile.txt", 2, True)

OpenAsTextStream
Синтаксис:
File.OpenAsTextStream([iomode, [format]])
где
iomode – режим открытия файла.
Принимает значения:
1 – файл открывается для чтения. Записывать в него нельзя.
2 – файл открывается для записи.
3 – файл открывается для добавления данных
format – кодировка для открытия файла.
Принимает значения:
- 2 – открывается, используя системную кодировку
- 1 – открывается в кодировке Unicode
0 – открывается в кодировке ASCII.
Примечание. Перед использованием метода **OpenAsTextStream** объект **File** должен быть создан методом **GetFile**.
После открытия файла и всех изменений он должен быть закрыт методом **Close**.
Запись в файл
Запись информации в текстовый файл производится с помощью методом **Write**, **WriteLine** и **WriteBlankLines**.
Методы:
Write - записывает данные в файл в одну строку
WriteLine – записывает данные, с переходом на новую строку.
WriteBlankLines – записывает пустую строку.
Например:

```
Set FSO = CreateObject("Scripting.FileSystemObject")
Set f = FSO.OpenTextFile("testfile.txt", 2, True)
'Открываем файл, если он не создан, создаем его
f.WriteLine "Файл создан VBScript!"
'записываем строку
f.WriteBlankLines(3)
'записываем 3 пустые строки
f.Write "Это снова я!"
'записываем строку
f.Close
'закрываем файл
```

Чтение из файла

Для чтения данных из файла используются методы **Read**, **ReadLine**, **ReadAll**, **Skip** и **SkipLine**.
Методы:
Read – читает из файла указанное количество символов.
ReadLine – читает строку из файла
ReadAll – считывает весь файл
Skip – пропускает указанное количество символов
SkipLine – пропускает строку

Пример:

```
Set FSO = CreateObject("Scripting.FileSystemObject")
 'Открываем файл, если он не создан, создаем его:
Set f = FSO.OpenTextFile("testfile.txt", 2, True)
f.WriteLine "Файл создан VBScript!"
'записываем строку
f.WriteBlankLines(3)
'записываем 3 пустые строки
f.Write "Это снова я!"
'записываем строку
f.Close
'закрываем файл
Set t_file = FSO.OpenTextFile("testfile.txt", 1)
'открываем файл для чтения
source = t_file.ReadAll
'читаем весь файл и передаем его переменной source
MsgBox source
'выводим содержание файла
t_file.Close
'закрываем файл
```

Работа с папками

Для работы с папками и их свойствами **FSO** предоставляет методы **Copy**, **Move**, **CreateFolder**, **Delete**, **DateCreated**, **DateLastAccessed**, **DateLastModified**, **Size**, **FolderExists**, которые работают аналогично файловым методам.
Например

```
Set FSO = CreateObject("Scripting.FileSystemObject")
 If Not FSO.FolderExists("Новая папка") Then
'если папка не существует
'создаем папку "Новая папка":
FSO.CreateFolder("Новая папка")
End If
Set f1 = FSO.GetFolder("Новая папка")
'получаем папку
Info = "Свойства папки: " + f1.Name + Chr(10) + Chr(10)
Info = Info & "Размер: " & f1.Size & Chr(10)
'получаем размер папки
Info = Info & "Создана: " & f1.DateCreated & Chr(10)
'время создания
Info = Info & "Изменена: " & f1.DateLastModified  & Chr(10)
'время последнего изменения
MsgBox Info
```

В результате выполнения получим на экране окно типа:

Диски

Сценарии не могут непосредственно работать с дисками в системе, поскольку объект **FileSystemObject** не предоставляет для этого методов. Но с помощью свойств объекта **Drive** и коллекции **Drives** мы можем получить некоторые сведения о дисках системы.

Для работы с диском необходимо как и при работе с файлами и папками создать объект с его свойствами с помощью метода **GetDrive**.

Синтаксис:

object.GetDrive drivespec

где

drivespec – имя диска. В кавычках располагается его имя. Это может быть "c" или "c:" или "c:\". При работе с сетевыми дисками это может быть "computer2\share1"

Пример:

```
Set FSO = CreateObject("Scripting.FileSystemObject")
'создаем объект FileSystemObject
Set Drv = FSO.GetDrive("c:")
'получаем диск C
```

DriveExists – выполняет проверку, существует ли указанный диск.

Пример:

```
Set fso = CreateObject("Scripting.FileSystemObject")
If fso.DriveExists("c") Then
MsgBox "Диск C присутствует"
Else
MsgBox "Диск C не найден"
End If
```

AvailableSpace – возвращает количество свободного места на диске, которым располагает пользователь, в байтах.

Синтаксис:

object.AvailableSpace

где

object - объект Drive

Следует заметить, что на дисках размером более 2 Гигабайт многие свойства, возвращающие пространство на диске могут работать неправильно, поскольку наибольшая величина целочисленного значения у языков сценариев не может быть больше чем 2 147 483 647, поэтому, если возвращенная величина будет больше, то сценарий возвратит некорректные данные.

FreeSpace – возвращает количество свободного места на диске.

TotalSize – всего места на диске.

DriveType – тип диска.

Возвращаемые значения свойства DriveType:

Значение	**Объяснение**
0	Тип не может быть определен
1	Сменный носитель или дисковод для гибких дисков
2	Обычный HDD
3	Сетевой диск
4	CD-ROM
5	Виртуальный RAM-диск

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

```
Set d = fso.GetDrive("c")
Select Case d.DriveType
Case 0: t = "Неопределен"
Case 1: t = "Дисковод 3.5'"
Case 2: t = "Обычный HDD"
Case 3: t = "Сетевой"
Case 4: t = "CD-ROM"
Case 5: t = "RAM Disk"
End Select
MsgBox "Диск " & "c" & ": - " & t
```

FileSystem – возвращает тип файловой системы FAT, NTFS, или CDFS.
Пример:

```
Dim FSO, D, FileSys
Set FSO = CreateObject("Scripting.FileSystemObject")
Set D = FSO.GetDrive("C")
FileSys = D.FileSystem
MsgBox "Файловая система на диске C: " + FileSys
```

IsReady – логическое значение. Возвращает true, если диск готов к использованию.
Пример:

```
Dim FSO, d, t
Set FSO = CreateObject("Scripting.FileSystemObject")
Set d = FSO.GetDrive("a")
If d.IsReady Then
MsgBox "Устройство готово к использованию"
Else
MsgBox "Устройство не готово "
End If
```

RootFolder – возвращает путь к корневому каталогу. Для диска C – "c:\", для a – "a:\" и т.д.
SerialNumber – серийный номер устройства в десятичной системе.
Пример:

```
Dim FSO, D, Serial
Set FSO = CreateObject("Scripting.FileSystemObject")
Set D = FSO.GetDrive("C")
Serial = D.SerialNumber
'получаем серийный номер диска C в десятичной системе
MsgBox "Файловая система на диске C: " & Hex(Serial)
'выводим его в шестнадцатеричной системе
```

ShareName – сетевой адрес устройства
VolumeName – возвращает/устанавливает метку диска
Синтаксис:
object.VolumeName [= newname]
где *object* - объект Drive
Необязательный параметр:
newname - новая метка диска
Пример:

```
Dim FSO, D, Vol
Set FSO = CreateObject("Scripting.FileSystemObject")
Set D = FSO.GetDrive("C")
Vol = D.VolumeName
If Vol = "" Then
'если метка не задана
```

```
MsgBox"Метка для диска C не задана"
Else
MsgBox"Метка диска C: " & Vol
End If
```

Доступ к файлам и папкам

Свойства **Name**, **ShortName**, **ParentFolder**, **Path** и **ShortPath** служат для определения имен и местоположения файлов и папок.
Свойство **Name** содержит обычное имя. А свойство **ShortName** - короткое имя т.е. имя в формате MS DOS.

```
Set FSO = CreateObject("Scripting.FileSystemObject")
'создаем FileSystemObject
FSO.CreateTextFile "Текстовый файл.txt", true
' создаем файл Текстовый файл.txt
Set file1 = FSO.GetFile("Текстовый файл.txt")
LongName = file1.Name
' получаем длинное имя файла
ShName = file1.ShortName
' получаем короткое имя
Names = "Имя: " + LongName + Chr(10) + "Имя DOS: "+ ShName
MsgBox Names, 64, "Файл: " + file1.Name
'выводим результат
```

Свойства **Path** и **ShortPath** указывают путь. **ShortPath** - путь в формате MS DOS.
На этот раз в примере используем папки.

```
Set FSO = CreateObject("Scripting.FileSystemObject")
FSO.CreateFolder("Новая папка")
'создаем папку
Set Folder1 = FSO.GetFolder("Новая папка")
LongPath = Folder1.Path
ShPath = Folder1.ShortPath
PathInfo = "Путь: " + LongPath + Chr(10) + "Путь DOS: " + ShPath
MsgBox PathInfo, 0, "Свойства: "  + Folder1.Name
```

Последнее свойство **ParentFolder** возвращает имя папки, которая содержит файл или папку.

```
Set FSO = CreateObject("Scripting.FileSystemObject")
Set file1 = FSO.GetFile(WScript.ScriptName)
'получаем файл текущего сценария
PFolder = file1.ParentFolder
MsgBox "Сценарий " + WScript.ScriptName + " находится в папке " + Pfolder
```

Атрибуты файлов и папок

Для изменения атрибутов файлов и папок служит свойство **Attributes**
Синтаксис:
object.Attributes = newattributes
где
object – объект **File** или **Folder**
newattributes – новые установленные атрибуты
Параметр *newattributes* может принимать следующие значения:

Константа	Значение	Действие	Описание
Normal	0	чтение/запись	Обычный файл без атрибутов

ReadOnly	1	чтение/запись	Только чтение
Hidden	2	чтение/запись	Скрытый
System	4	чтение/запись	Системный
Volume	8	только чтение	Метка диска
Directory	16	только чтение	Папка
Archive	32	чтение/запись	Архивный
Compressed	128	только чтение	Сжатый файл

Чтобы установить или снять атрибут с файла, нужно соответственно прибавить или удалить нужное значение.

Type

Метод **Type** возвращает тип файла. При применении его к объекту **Folder** – возвращает значение "Папка с файлами".

```
Dim FSO, D
Set FSO = CreateObject("Scripting.FileSystemObject")
Set F = FSO.GetFile(WScript.ScriptName)
MsgBox "Тип файла "+ F.Name + " : " +F.Type
```

Если файл сценария сохранен с именем tst.vbs, то при выполнении получим сообщение "Тип файла u:\tst.vbs : Файл сценария VBScript"

Использование XML

WSF-файлы

В предыдущих разделах для использования сценариев мы пользовались файлами того типа, какой язык сценариев применялся: VBScript или JScript, то есть файлами js и vbs.
В Windows Scripting Host в версии 2.0 появилось возможность использовать файлы в которых мы можем использовать эти два языка одновременно. Это файлы WSF.
WSF-файлы (Windows Script File) представляют собой файлы с разметкой XML. Те, кто имел дело с HTML файлами, вероятно слышал об XML.
XML – разметка, подобна разметке HTML, с то только разницей, что HTML – отвечает за размещение информации на экране, а XML – за хранение информации.
Перед обычными файлами сценариев vbs и js, WSF имеет большое преимущество:
1. В одном wsf-файле можно совмещать сценарии, написанные на разных языках (и не только VBScript и JScript).
2. Кроме того в одном wsf-файле возможно хранить множество совершенно разных сценариев. Т.е. вы можете хранить сценарии, написанные на каком-либо языке или сразу на нескольких языках сценариев, в любом количестве, и все эти сценарии хранить в одном wsf-файле
Всего в wsf-файлах могут применяться 8 XML-элементов. В следующей главе мы рассмотрим их.

Элементы XML

XML хотя и имеет много общего с HTML представляет собой язык более сложный и изучение всех его элементов и технологий, которые могут использоваться в wsf-файлах выходит за рамки нашей книги.
Это не должно никого смущать, так как для эффективного использования wsf-файлов, и применения всех полученных до сих пор навыков, достаточно знания всего нескольких XML-элементов.

<script>

Элемент <SCRIPT> используется для вставки сценария в wsf-файл. То есть сам сценарий всегда будет располагаться в элементе <SCRIPT>.
Синтаксис:

<script language="language" [src="strFile"]>
сценарий
</script>
где
language – используемый язык сценария. Для VBScript это "VBScript" или "VBS", для JScript это соответственно "JScript ".
Необязательный параметр:
strFile – задается, если предполагается, что в данном сценарии будут использованы сценарии из других vbs и js-файлов.
<job>
В элементе **<job>** (задание) располагаются элементы <script>. Таким образом в пределах элемента <job> будет находиться отдельный сценарий.
Синтаксис:
<job [id=JobID]>
Отдельный сценарий
</job>
где
JobID – это его имя, по которому его можно вызвать из wsf-файла.
В следующем примере в "задании" VBS_job располагается сценарий, написанный на VBScript, выводящий надпись "Используется VBScript":

```
<job id="VBS_job">
<script language="VBScript">
WScript.Echo "Исполользуется VBScript"
</script>
</job>
```

Подобных "заданий" в wsf-файле может быть сколько угодно, и все они могут содержать сценарии на разных языках.

<package>
Как элемент <job> может объединять несколько элементов <script>, так и элемент <package> служит для объединения элементов <job> (от одного до практически любого их количества).
Синтаксис:

```
<package>
одно или несколько заданий
</package>
```

Вышеперечисленных элементов обычно хватает для полноценной работы с wsf-файлами.
Давайте теперь научимся вызывать сценарий из wsf-файла.
Для этого существует несколько способов:
1.Если файл содержит один сценарий (одно задание(<job>)), то для запуска его нужно запустить обычным способом, так же как и vbs и js-файлы. Это при условии, что задание не будет иметь своего ID.
Пример:

```
<job>
<script language="JScript">
WScript.Echo("Hello World!!!");
<script>
</job>
```

2. Если файл содержит несколько заданий, то любое из них можно вызвать, с помощью ключа запуска //Job:<JobID>.
Например, создадим wsf-файл, содержащий два задания VBS_job и Js_job, содержащие сценарии на разных языках:

```
<package>
```

```
<job id="VBS_job">
 <script language="VBScript">
WScript.Echo "Это сценарий VBScript"
 </script>
 </job>
 <job id="JS_job">
 <script language="JScript">
 WScript.Echo("Это сценарий JScript");
 </script>
 </job>
 </package>
```

Назовем его 2in1.wsf. Попробуем запустить его щелчком мыши. И получим результат:
Когда wsf-файл запускается без параметров, то WSH выполняет первое идущее задание. В данном случае это VBS_job. Чтобы вызвать задание JS_job, используем параметр командной строки //job (см. гл "Запуск сценариев"). Наберем в командной строке:
Wscript //job:JS_job 2in1.wsf
Получим:
Для запуска задания VBS_job, нужно его id подставить в ключ "//job:"
Как уже говорилось важнейшей особенностью wsf-файлов является возможность объединения в одном файле сценариев, написанных на разных языках.
Ранее для вызова из сценария JScript окна ввода текста, вызывали его из другого файла VBScript с использованием переменных окружения. Теперь же для решения данной задачи можно поместить оба сценария в один файл и не использовать переменные окружения:

```
 <package>
 <job>
 <script language = "VBS">
Sub EnterName()
Name = InputBox("Введите свое имя","")
MsgBox "Привет " + Name
End Sub
 </script>

 <script language="JScript">
//вызываем подпрограмму EnterName():
EnterName()
</script>
</job>
</package>
```

Также в wsf-файлах допускается использование внешних файлов. Изменим предыдущий сценарий, где будем вызывать подпрограмму EnterName() из другого файла. Достигается это использованием атрибута src.
Создадим файл, содержащий подпрограмму EnterName() MySub.vbs:

```
Sub EnterName()
Name = InputBox("Введите свое имя")
MsgBox "Привет " + Name
End Sub
```

Теперь напишем wsf-файл который будет его использовать:

```
 <Job>
 <script language="VBScript" src="MySub.vbs"/>
 <script language="JScript">
 EnterName();
```

</Script>
</Job>

Другие элементы XML

Нами были рассмотрены основные элементы, без которых не обойтись при использовании wsf-файлов. Рассмотрим остальные элементы.

<?xml ?> - данный элемент объявляет об использовании в файле языка XML. Он не имеет конечного тега и должен располагаться в начале файла.

Синтаксис:

<?XML version="version" [standalone="DTDflag"] ?>

где

version - используемая версия XML. На сегодняшний момент это 1.0

Необязательный параметр:

DTDflag - логический параметр, показывающий включает ли документ ссылку на DTD (Document Type Definition).

<?job ?> - определяет информацию об атрибутах обработки ошибок.

Синтаксис:

<?job error="flag" debug="flag" ?>

где

error и *debug* - логические переменные, указывающие как обрабатывать ошибки сценария. По умолчанию принимают значения false.

<object> - служит для создания объектов.

<reference> - предоставляет доступ к константам, во внешних по отношению к объектах.

<resource> - содержит символьные или числовые данные, которые по каким-либо причинам не нужно вставлять в сценарий.

Обработка ошибок

On Error Resume Next - игнорирование ошибок сценариев.

Обратите внимание, что оператор **On Error Resume Next** при исполнении сценария будет игнорировать ошибки выполнения сценария, а не синтаксис языка. Т.е. если, например, будет вызвана несуществующая процедура или будет невозможно создать файл - сценарий не будет выдавать сообщение об ошибке, а будет выполнять последующие операторы. Но если будут

ошибки синтаксиса в конструкциях языка, например, отсутствовать закрывающие скобки или кавычки - появится сообщение об ошибке.

Примеры работы с файлами и директориями. Обход директорий

Выведем список файлов указанной директории (в данном примере в качестве текущей директории выбрана корневая директория "c:\") в MsgBox:

```
set fso=Createobject("Scripting.FileSystemObject")
set cur_dir = fso.GetFolder("C:\")
ttt="Список файлов директории"+cur_dir.path+chr(13)
for each fil in cur_dir.files
ttt = ttt + fil.path+chr(13)
next
msgbox ttt
```

Выведем список поддиректорий указанной директории (в данном примере в качестве текущей директории выбрана корневая директория "c:\") в MsgBox:

```
set fso=Createobject("Scripting.FileSystemObject")
set cur_dir = fso.GetFolder("C:\")
ttt=ttt+chr(13)+chr(13)+"Список поддиректорий "+ cur_dir.path +chr(13)
for each sf in cur_dir.subfolders
ttt = ttt + sf.path + chr(13)
next
msgbox ttt
```

Выведем список файлов и список поддиректорий указанной директории (в данном примере в качестве текущей директории выбрана корневая директория "c:\") в MsgBox используя пользовательские функции. Обратите внимание на явное указание способа передачи параметров функции («по ссылке» – ByRef):

```
set fso=Createobject("Scripting.FileSystemObject")
set cur_dir = fso.GetFolder("C:\")
ttt = ""
i=list_file(ttt,cur_dir)
i1=list_dir(ttt,cur_dir)
msgbox ttt

function list_file(ByRef txt ,ByRef fldr)
txt= txt + "Список файлов директории C: "+chr(13)
for each fil in fldr.files
txt = txt + fil.path+chr(13)
next
end function

function list_dir(ByRef txt,ByRef fldr)
ttt=ttt+chr(13)+chr(13)+"Список поддиректорий C:\ "+chr(13)
for each sf in fldr.subfolders
ttt = ttt + sf.path + chr(13)
next
end function
```

В заключение рассмотрим алгоритм обхода дерева состоящего из директорий, поддиректорий и файлов файловой системы. Выведем список файлов всех поддиректорий, начиная с указанной директории (в данном примере в качестве текущей директории выбрана корневая директория "c:\") в MsgBox. Для решения этой задачи используем рекурсивную функцию list_dir. Эта функция в качестве параметра получает объект - текущую директорию. После вызова функции сначала выводится список файлов текущей директории, а затем – для каждой поддиректории этой директории, вновь вызывается та же самая функция. Процесс продолжается до тех пор, пока не будут обработаны все поддиректории.

```
set fso=Createobject("Scripting.FileSystemObject")
set cur_dir = fso.GetFolder("C:\inetpub")
ttt = ""
i1=list_dir(ttt,cur_dir)
msgbox ttt

function list_dir(ByRef txt,ByRef fldr)
n=0
txt = txt+"Директория: "+fldr.path + ": "
for each fil in fldr.files
n=n+1
next
txt = txt + " - "+CStr(n)+" файлов"+chr(13)
for each sf in fldr.subfolders
res=list_dir(ttt,sf)
next
end function
```