

# Первые шаги

<http://www.firststeps.ru/>

Всё для начинающих программистов. На сайте вы найдёте информацию по следующим темам:

- TASM
- Html&WEB
- PHP
- VRML
- Java
- CGI
- Perl
- Linux
- Visual C++
- Справка WinAPI
- Visual Basic
- Visual FoxPro
- C Sharp & .NET
- C++ Builder
- VBA by Step
- 1C
- GIS
- SQL by Step
- Алгоритмы
- Безопасность
- Протоколы
- MS Office

## Авторы сайта

### **Каев Артем Геннадьевич (E-mail)**

Родился: 13 октября 1972 года.

Московский геологоразведочный техникум. Московская Государственная Геологоразведочная Академия (бывший МГРИ), горный инженер геолог, специализация геоинформатика, математические методы в геологии. В 1996 - 1997 служба в армии. Одиннадцатая отдельная воздушно-десантная бригада. г.Улан-Удэ.



Специализированное обучение: разработка приложений - VC++, геоинформационные системы - ArcInfo.

Верхне-Индигорская геологоразведочная экспедиция, руководитель ВЦ.

С 1 января 2002 года, начальник ГИЦ Верне-Индигорской экспедиции

Увлечений практически нет. Работа и сайт - Первые шаги.

---

### **Кузин Андрей Владимирович (E-mail)**

Родился: 9 апреля 1980 года.

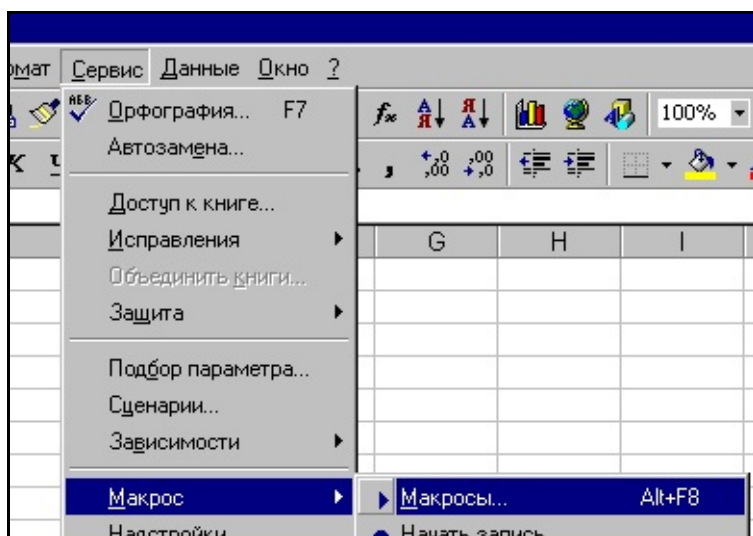


Московский Институт Электронной  
Техники - МИЭТ. Специализация -  
"Организация и технология защиты  
информации". Защитил диплом на  
отлично 7 июня 2002 года, тема  
"Контроллер защищенного канала  
передачи данных в сети Ethernet".

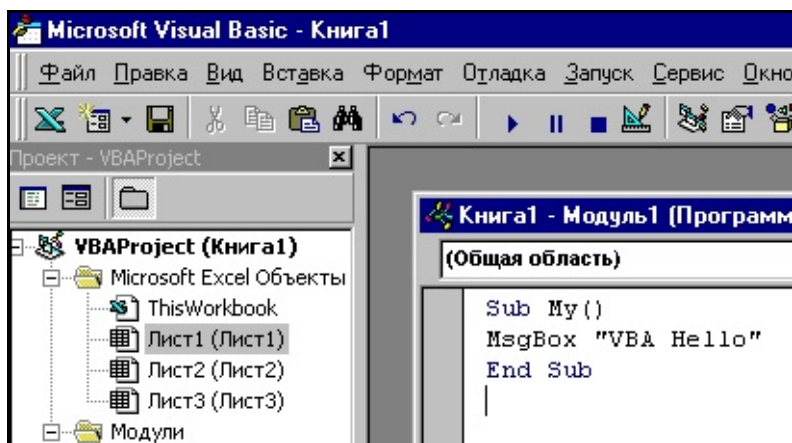
Вебмастер сайта "Первые Шаги".  
Системный администратор сети МЖК  
"Зеленоград".

# Шаг 1 - Первый макрос

Создаются макросы в меню **Сервис - Макрос - Макросы (Alt-F8)**:



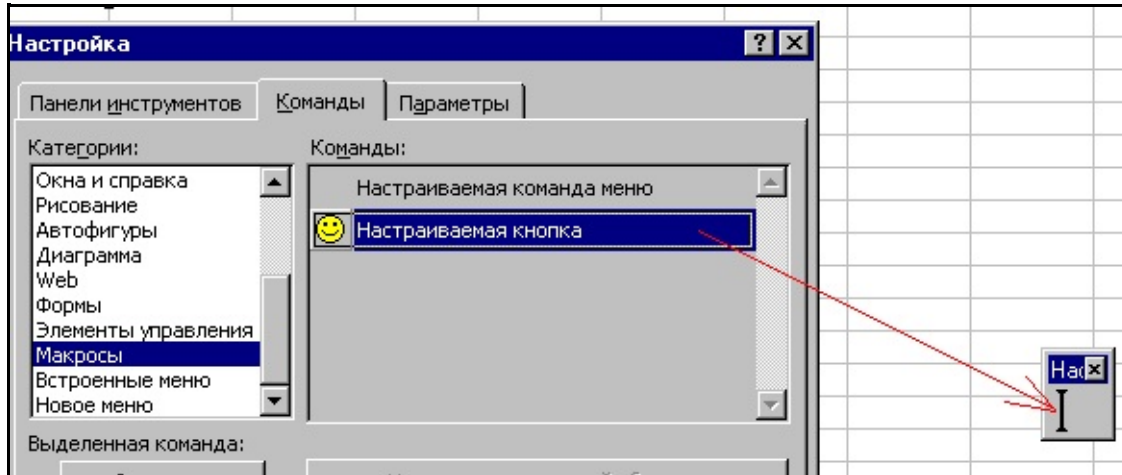
Появится табличка "макросы". В поле имя введите "MY", а место нахождения в поле "Находится" выберите - "Это книга". Кнопка "Создать" станет активной нажмите её. Появится редактор **VBA** введите код, как на рисунке ниже.



Здесь используется функция **MsgBox**, которая выводит на экран окно сообщения. Закройте редактор **VBA** файл - закрыть.

Дальше мы привяжем макрос к кнопке. Для этого создадим свою панель Инструментов. **Вид - панели инструментов - Настройка**. Нажмите

создать и у Вас появится панель инструментов настраиваемая 1. Теперь перейдем к вкладке команды в категории выбираем макросы. Хватаем веселую желтую рожицу и тащим на панель.



Теперь на рожице нажимаем правой кнопкой мыши и выбираем пункт меню "назначить макрос". Выбираем наш макрос. Нажимаем **Ок** и закрываем окно настройки. Теперь можно испытать. Нажмите кнопку, макрос выполняется и появляется надпись.

Классно, работает.

---

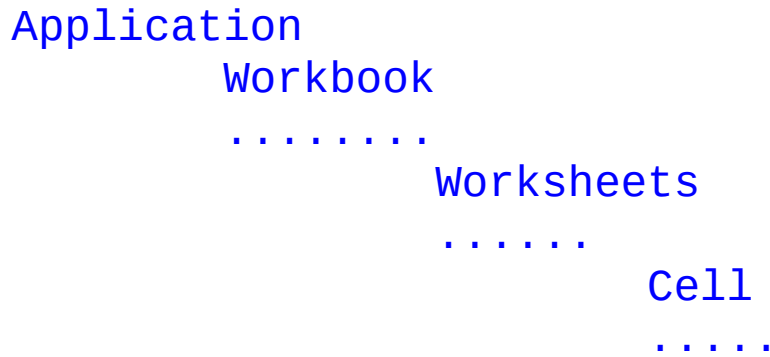
[Следующий Шаг](#) Автор [Каев Артем](#).

## Шаг 2 - Объектная модель Excel

Мы будем изменять наш макрос, зайдите в пункт меню "макросы", выберите наш и скажите "изменить":

```
Sub Test()  
    Dim book As String  
    Dim sheet As String  
    Dim addr As String  
    addr = "C"  
    book = Application.ActiveWorkbook.Name  
    sheet = Application.ActiveSheet.Name  
  
    Workbooks(book).Activate  
    Worksheets(sheet).Activate  
  
    Range("A1") = book  
    Range("B1") = sheet  
  
    Dim xList As Integer  
    xList = Application.Sheets.Count  
    For x = 1 To xList  
        Dim s As String  
        s = addr + LTrim(Str(x))  
        Range(s) = x  
    Next x  
End Sub
```

Программирование на **VBA** можно рассматривать, как управление объектами приложения. Вот именно объектами и управляет наше приложение. В нашем случае, если упростить иерхическую архитектуру, то это выглядит так.



То есть главный объект - приложение. В приложении могут быть несколько книг ( **Workbook** ), внутри которых находятся листы ( **Worksheets** ) и листы разбиты на ячейки ( **Cell** ). При работе активными могут быть только одна книга и один лист. Вот я своим макросом и пытаюсь это выяснить. А заодно сколько листов в текущей книге.

**DIM** - объявляет переменную с типом **string**. Используя объект **Application**, мы получаем имена текущих книг и листа. С помощью **Range("...")** можно выделить ячейку и поместить значения в неё или считать. Вообще объекты имеют огромное количество свойств. Задача программиста на **VBA** знать эти свойства и методы. Ну я думаю мысль этого шага понятна :-))

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)  
Автор [Каев Артем](#).

## Шаг 3 - Коллекции в VBA

В любом языке программирования массивы (коллекции) занимают большое место. При этом именно понятие коллекция принимает широкое распространение. В **Visual C++** понятие коллекция аналогична шаблону. При программировании на **VBA** понятие коллекции приобретает большой смысл. Коллекции встречаются на каждом шагу.

Обычно коллекции имеют 4 метода:

- Add
- Remove
- Count
- Item

Было бы наверно логично подумать, что в **MS OFFICE** есть специализированный класс коллекций, на подобии шаблонов в **C++**. Но это не так. Для каждого типа объектов объявляется своя коллекция. Так же есть некоторые коллекции, которые отличаются названием методов. Всё это немного странно, но что сделаешь :-)

Для понимания работы с коллекциями создадим имитирующий коллекцию книг в **Excel**:

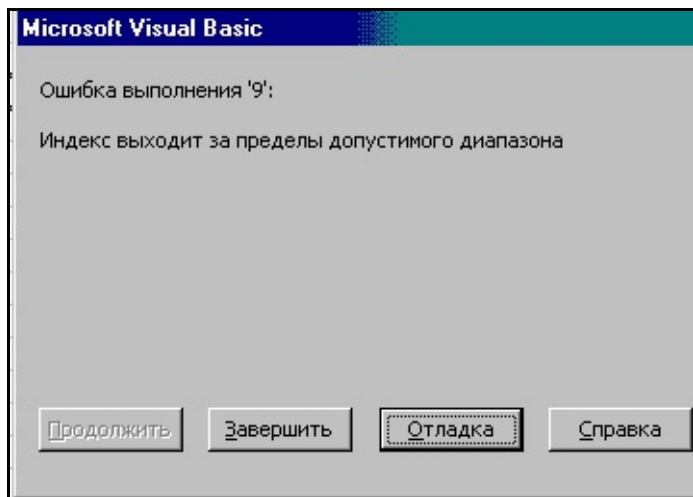
```
Sub Test()  
    Dim MyCollection As New Collection  
    With MyCollection  
        .Add ("Книга 1")  
        .Add ("Книга 2")  
        .Add ("Книга 3")  
        MsgBox (Str(.Count))  
        MsgBox (.Item(1))  
        .Remove (1)  
        MsgBox (.Item(1))  
    End With  
End Sub
```



## End With End Sub

Первой строкой мы создаем переменную типа коллекция. Далее мы используем оператор **With**, чтобы не использовать многократно **MyCollection** и тем самым сократить код. Методом **Add** мы добавляем данные в коллекцию. **Count** возвращает Вам количество элементов в коллекции. **Item** возвращает элемент коллекции, а **Remove** удаляет по индексу.

Метод **Remove** опасен тем, что он может выйти за пределы массива и вы получите подобную ошибку.



Обработку ошибок мы рассмотрим позже.

Заканчивая краткое описание коллекций следует отметить, что они могут содержать и элементы разных типов, что отличает их от массивов. Размер коллекции динамически изменяется. При удалении дырок не обнаруживается. Лафа вообще после C++ :-))

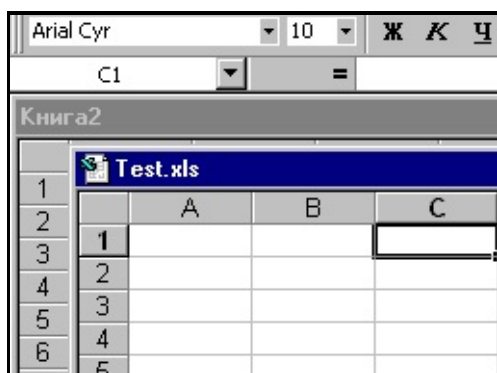
---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 4 - Коллекция Workbooks в Excel

Итак, в **Excel** самую верхушку составляет объект **Application**. Это объект приложение. И этот объект содержит ряд коллекций. Первая коллекция это коллекция рабочих книг **Workbook**. Вот как это выглядит на экране (визуализуется). То что есть внизу на экране и есть отображение коллекции книг.



Естественно вы как программист должны уметь со всем этим работать программным путем. Определять какие книги загружены, добавлять и удалять, и делать еще много вещей. При этом без меню и мышки программно.

Первая коллекция это коллекция книг. Первое, что нужно узнать это сколько книг открыто. Вот как это сделать.

```
Sub Test()  
    MsgBox (Str(Application.Workbooks.Count))  
End Sub
```

Функция **Str** переводит число в строку. Метод **Count** Вам известен. Он возвращает количество элементов коллекций. Как и на рисунке сверху у меня две книги, о чем и сообщит мне окно сообщения.

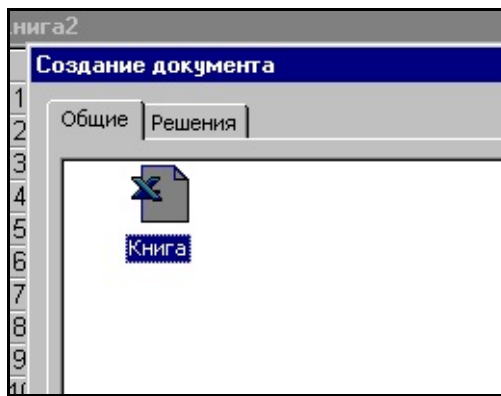
Данная коллекция обладает некоторой спецификой. Это связано с тем, что книги могут храниться в файлах. Поэтому при работе с этой коллекцией предусмотрено две функции добавления **Add** и **Open**:

## Add(template)

Добавляется книга на основе некоторого шаблона. Шаблоном может выступать, как настоящий шаблон с расширением **xlt**, так и просто файл **xls**. Смотрим пример:

```
Sub Test()  
    Application.Workbooks.Add ("Книга"  
End Sub
```

Откуда я взял имя шаблона Книга? Вот смотрите сами.



А здесь как он оказался. Молодцы !!! Честное слово. Это важный вопрос. Сначала, что такое шаблон. Шаблон это специальная рабочая книга, образец для создания документов. Тип файлов шаблонов **xlt**, то есть такое расширения носят шаблоны. Что задается в шаблоне ?

- Форматы ячеек
- Пользовательские меню, макросы и панели инструментов
- номера и типы листов
- стили строк и колонок
- текст, даты формулы . можно некоторые константы и графика.

Когда на основе шаблона создается документ , это просто копия шаблона, полная копия. Имея созданную книгу её можно сохранить как шаблон. Для этого используйте пункт меню "Сохранить как тип шаблона". Конечно шаблоны хранятся в некоторой папке, путь к ней указан в реестре:

[HKEY\\_CURRENT\\_USER\Software\Microsoft\Office](#)

Вы можете поменять этот тип на сетевой для одновременного изменения шаблона на всех компьютерах в случае, например, изменения телефона вашей фирмы.

Следующий метод **Open**, у которого куча параметров. Но единственный важный это имя файла, остальные можно опустить.

```
Sub Test()  
    Application.Workbooks.Open ("c:\1\  
End Sub
```

Приведу я параметры на всякий случай:

```
expression.Open(FileName, UpdateLinks, Read  
    Password, WriteResPassword, IgnoreRe  
    Origin, Delimiter, Editable, Notif
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 5 - Далее про Workbooks в Excel

В прошлый раз мы научились добавлять книги в коллекцию. Теперь научимся удалять и получать информацию о книгах. Так как книга ассоциируется с файлом, то и метод удаления книги из коллекции называется **Close**. Но для удаления необходимо получить доступ к элементу.

```
Sub Test()  
    Application.Workbooks.Item(1).Close  
End Sub
```

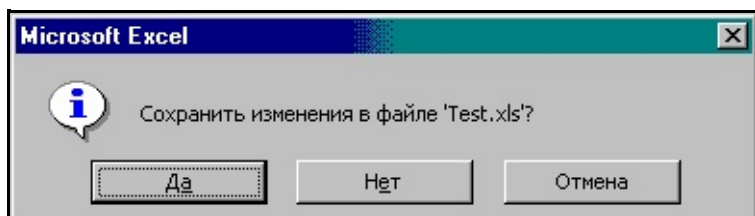
В таком варианте закроются все книги:

```
Sub Test()  
    Application.Workbooks.Close  
End Sub
```

Функция **Close** имеет ряд необязательных параметров. Вот они.

```
expression.Close(SaveChanges, FileName, Ro
```

Первый параметр **SaveChanges** типа **BOOL**, если установить **TRUE** сделанные изменения сохранятся, в противном случае нет. Если параметр упускается, то при закрытии появляется диалоговое окно с вопросом о необходимости сохранения.



Следующий параметр **FileName** необходим, когда идет вопрос о закрытии книги не связанной еще с именем файла. Последний параметр связан с одновременной работой над книгой. Он типа **BOOL**.

Получить доступ к книгам в коллекции можно используя метод **Item()**:

```
Sub Test()  
    MsgBox (Application.Workbooks.Item  
    MsgBox (Application.Workbooks.Item  
End Sub
```

Как видите, доступ можно получить по индексации и по имени книги, следует знать, что имя книги это имя файла, в котором он хранится. Получив доступ по индексу в первой строке я узнал имя. А во второй по имени получил доступ к свойствам объекта **Workbook** и получил полное имя, название файла и путь, который и вывел на экран в виде сообщения.

Еще одно свойство - это создатель книги, и называется он **Creator**

```
Sub Test()  
    If (Application.Workbooks.Creator :  
        MsgBox "Excel Creator"  
    Else  
        MsgBox "Not Excel Creator"  
    End If  
End Sub
```

С помощью **Parent** можно получить доступ к старшему объекту коллекции при выполнении кода изложенного ниже, появится **MS Excel**, ну а Вы что подумали ? :-)

```
Sub Test()  
    MsgBox (Application.Workbooks.Pare  
End Sub
```

---

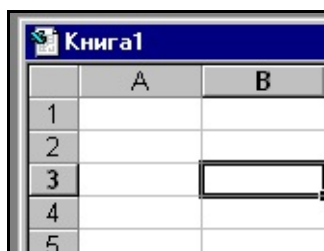
[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 6 - Имена ячеек и адресация в Excel

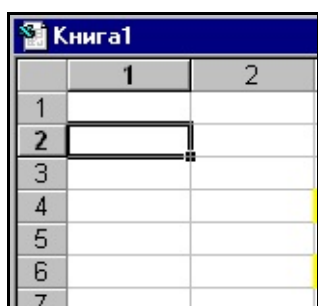
Раз мы че-то задумали программировать нужно разобраться с тем, как **Excel** производит адресацию ячеек и как можно им давать имена. По умолчанию используется стиль **A1**. Это когда по строкам используется алфавит, а по горизонтали цифры. Например **D10** это десятая строка в колонке **D**. Есть и стиль называемый **R1C1**, который наиболее полезен при вычислении позиции строки и столбца в макросах, а также при отображении относительных ссылок. В стиле **R1C1**, после буквы "**R**" указывается номер строки ячейки, после буквы "**C**" номер столбца.

Стиль **A1**:



	A	B
1		
2		
3		
4		
5		

Стиль **R1C1**:

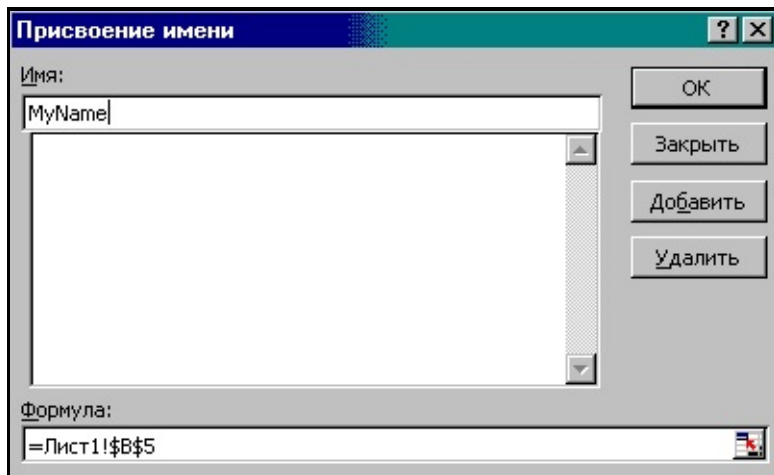


	1	2
1		
2		
3		
4		
5		
6		
7		

При работе стили переключаются в меню **Сервис -> Параметры -> Общие -> Стиль ссылок**, при реальном программировании наиболее удобно пользоваться не этими стилями, а именами ячеек. Тогда работа с вашей ячейкой похожа на работу с обычной переменной. Что многим более привычно и удобно. Например для констант или полей форм.

Для того, чтобы дать имя ячейке наведите на неё курсор. Выберите меню **Вставка -> Имя -> Присвоить**. Появится диалоговое окно, куда надо

ввести имя и нажать кнопку **ОК**.

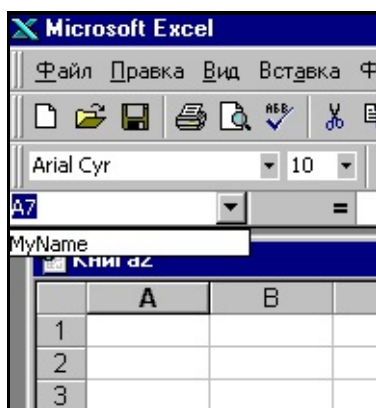


После присваивания имени вы введете число в эту ячейку, а в другой создайте формулу:

**=MyName+10**

Данная запись намного информативнее, кроме того вы можете не заботиться о местоположении имени в таблице, можете менять его местоположение не заботясь о том, что Ваши формулы будут изменены. А особенно это важно при программировании. Эта мелочь позволит избежать Вам сложной адресации и отслеживания данных.

Узнать все имена можно здесь:



И здесь так же можно быстро переместиться к ячейке с заданным именем. Выберите её из списка и где бы она не находилась Вы окажетесь там :-)



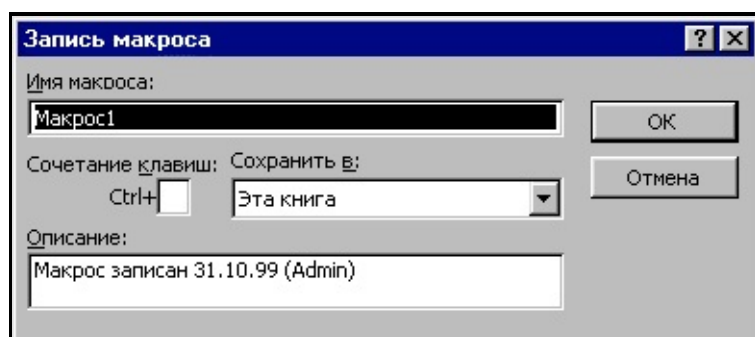
---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 7 - Запись макросов и что это дает

Попробуем записать макрос. Для этого выбираем пункт меню **Сервис** -> **Макрос** -> **Начать запись**, в ответ на это Вы получите следующее диалоговое окно.



Здесь вы можете указать название макроса, быструю клавишу, где хранить макрос. Оставьте все как есть и нажмите кнопку **ОК**. В результате у вас появится значек, который говорит о том, что идет запись. Вообще при записи макросов рекомендуется пользоваться клавишами, но я, например, и мышкой пользуюсь и записывается. Итак, появится значек.



А теперь выполните следующие действия. Создайте новую книгу, введите два числа в колонку, примените автосуммирование, сохраните книгу. После чего остановите запись макроса нажав на эту квадратную синюю кнопку. Зайдите в меню **Сервис** -> **Макрос** -> **Макросы**, у вас в диалоговом окне появится название вашего макроса. Выделите его мышкой и нажмите **Изменить**. Должен появиться такой код:

```
Sub Макрос1()  
'  
' Макрос1 Макрос
```

```
' Макрос записан 31.10.99 (Admin)
'
Application.WindowState = xlMinimized
Application.WindowState = xlNormal
Workbooks.Add
ActiveCell.FormulaR1C1 = "12"
Range("A2").Select
ActiveCell.FormulaR1C1 = "23"
Range("A3").Select
ActiveCell.FormulaR1C1 = "=SUM(R[-:
Range("A4").Select
ChDir "C:\WINDOWS\Рабочий стол"
ActiveWorkbook.SaveAs FileName:="C
                FileFormat:=xlNormal, Pass
                ReadOnlyRecommended:=False

End Sub
```

Да Вы не ошиблись это код **VBA**. Этот код ваших операций. Конечно здесь нет циклов и массивов. Но здесь есть решение задачи. Если вы знаете как сделать в ручную, но не знаете как запрограммировать, запишите макрос, добавьте функциональность за счет выбора и циклов, продумайте адресацию. Но общая стратегия у Вас есть. Кроме того, если вы хотите запрограммировать, например, открытие файла **DBF** в **Excel**, то чего гадать с параметрами. Запишите макрос и посмотрите.

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 8 - Коллекция Sheets

Данная коллекция представляет собой коллекцию листов (**Sheets**) в книге (**WorkBook**). Первое, что мы с Вами сделаем это получим количество листов в книге.

```
Sub Test()  
    MsgBox (Str(Application.Workbooks.Count))  
End Sub
```

Но под листом понимается не только клетки, но и диаграмма. Так же как и лист для расчетов диаграмма будет включена в подсчет листов. Как посмотреть имена листов. Просто. Есть свойство **Name**:

```
Sub Test()  
    With Application.Workbooks.Item("Тест")  
        For x = 1 To .Sheets.Count  
            MsgBox (Sheets.Item(x).Name)  
        Next x  
    End With  
End Sub
```

А как же лист с формулами отличить от диаграммы ? Попробуйте так. **Type** вернет Вам тип. Только не знаю документированный способ это или нет.

```
Sub Test()  
    With Application.Workbooks.Item("Тест")  
        For x = 1 To .Sheets.Count  
            MsgBox (Sheets.Item(x).Type)  
            If Sheets.Item(x).Type = xlTable Then  
                MsgBox Sheets.Item(x).Name  
            End If  
        Next x  
    End With
```

## End Sub

К коллекции листов есть возможность добавлять свои листы, для этого существует метод **Add**. Этот метод требует **4** параметра **Add(Before, After, Count, Type)**. Все эти параметры необязательные. Первые два отвечают за место вставки листа. Далее количество вставляемых листов **Count** и тип листа. Типы могут быть, например, такие. **xlWorksheet** для расчетного листа, **xlChart** для диаграммы. Если местоположение не указывать, то лист будет вставляться относительно текущего листа.

```
Sub Test()  
    With Application.Workbooks.Item("Т  
        Sheets.Add  
    End With  
End Sub
```

Метод **Parent** позволяет узнать какой книге принадлежит текущий лист.

```
Sub Test()  
    With Application.Workbooks.Item("Т  
        MsgBox (Sheets.Parent.Name  
    End With  
End Sub
```

Если у Вас есть желание, то некоторые листы можно убрать из обзора. Это бывает полезно, если у Вас есть константы или расчеты, которые Вы не хотите чтобы видели на экране в виде листов. Для этого можно использовать метод **Visible**. Устанавливая это свойство в **TRUE** или **FALSE** вы сможете убирать и показывать лист.

```
Sub Test()  
    With Application.Workbooks.Item("Т  
        .Sheets.Item("Лист5").Visible = Fa  
    End With  
End Sub
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 9 - Еще о Sheets

Один из полезных методов это метод **Copy**. Он позволяет создавать новый лист на основе существующего, то есть использовать лист как шаблон для других листов. Переименуйте любой лист в имя **Test**. Это можно сделать нажав правую кнопку мыши на названии листа и выбрав пункт меню **Переименовать**. Создайте на листе любое форматирование. **after** это лист, после которого произойдет вставка.

```
Sub Test()  
    With Application.Workbooks.Item("Т  
        Sheets("Test").Copy , afte  
    End With  
End Sub
```

У метода **Copy** есть особенность. Если не указывать параметры, то будет создана новая книга с копируемым листом.

```
Sub Test()  
    With Application.Workbooks.Item("Т  
        Sheets("Test").Copy  
    End With  
End Sub
```

При необходимости передвинуть лист есть метод **Move**:

```
Sub Test()  
    With Application.Workbooks.Item("Т  
        Sheets("Test").Move , afte  
    End With  
End Sub
```

Так как коллекция эта содержит объекты листа у неё есть несколько

полезных методов. Один из них **PrintPreview** позволяющий вызывать предварительный просмотр.

```
Sub Test()  
    With Application.Workbooks.Item("Т  
        Sheets("Test").PrintPreview  
    End With  
End Sub
```

Программным путем можно и выделять листы. Это метод **Select**. У него один параметр типа **BOOL**, если он установлен в **TRUE**, то происходит выделения листа, а если **FALSE**, то выделение объединяющее. Выделите другой лист и запустите следующий макрос.

```
Sub Test()  
    With Application.Workbooks.Item("Т  
        Sheets("Test").Select (Fal  
    End With  
End Sub
```

---

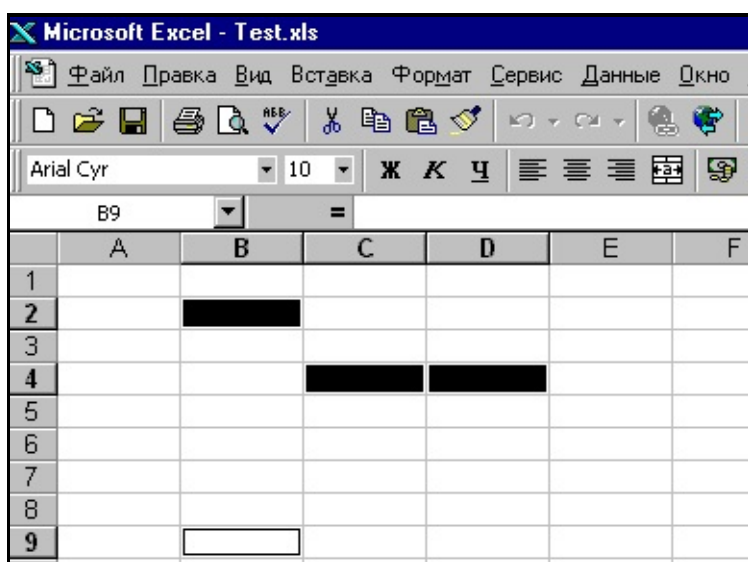
[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).



## Шаг 10 - Использование Range

Мы с вами будем использовать это свойство для выделения ячеек. Но прежде давайте просто посмотрим, как вообще можно их выделять. Конечно, если вы установите курсор на любую ячейку это то же выделение. Вы можете вводить в неё формулы или числа, менять формат. Выделить можно и несколько ячеек. Если установить курсор в одну ячейку и не отпуская левую кнопку мыши тащить, то выделится целый диапазон. Так же можно выделять отдельные ячейки, как на рисунке ниже.



Реализуется это довольно просто. Выделяете первую ячейку, держите клавишу **Ctrl** и не отпуская её другие ячейки. С выделенными диапазонами можно проводить большое количество операций. Вот для программной реализации этих возможностей и служит свойство **Range**. Оно есть у очень многих объектов **Excel**.

Этот пример показывает как выделить ячейку и поместить туда число.

```
Sub Test()
```

```
    With Application.Workbooks.Item("Т  
        Worksheets("Лист2").Active  
        Range("A2") = 2  
        Range("A3") = 3
```

```
End With
End Sub
```

Следует обратить внимание, что для нормальной работы этого метода рабочая книга и лист должны быть активными, иначе возникнет ошибка. Кроме того, наверно, надо перед каждым его использованием указывать книгу и лист, с которым работаем, дабы не внести изменения в другой, чему конечно пользователь или Вы будете рады...

С помощью этого метода можно помещать и формулы:

```
.....
Range("A4") = "=A2+A3"
.....
```

Можно указать и целый диапазон:

```
.....
Range("A2:A5") = 2
.....
```

Если вам захочется наоборот получить из ячейки формулу или значение, то Вам ни кто не мешает сделать вот так:

```
Sub Test()
    With Application.Workbooks.Item("Т
        Worksheets("Лист2").Active
        Range("A2") = 2
        Range("A3") = "=A2+2"
        MsgBox Range("A3").Formula
    End With
End Sub
```

Следующий пример очень важен, он показывает возможности абсолютной и относительной адресации. Мы создаем объект типа **Range**, а на основе

его производим адресацию. В следующем примере число поместится не в **A1**, а в **D3**. То есть у нас есть возможность выделять диапазон по абсолютному адресу, а внутри его использовать относительную адресацию.

```
Sub Test()  
    With Application.Workbooks.Item("Т  
        Worksheets("Лист2").Activa  
        Dim HelloRange As Range  
        Set HelloRange = Range("D3  
        HelloRange.Range("A1") = 3  
    End With  
End Sub
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

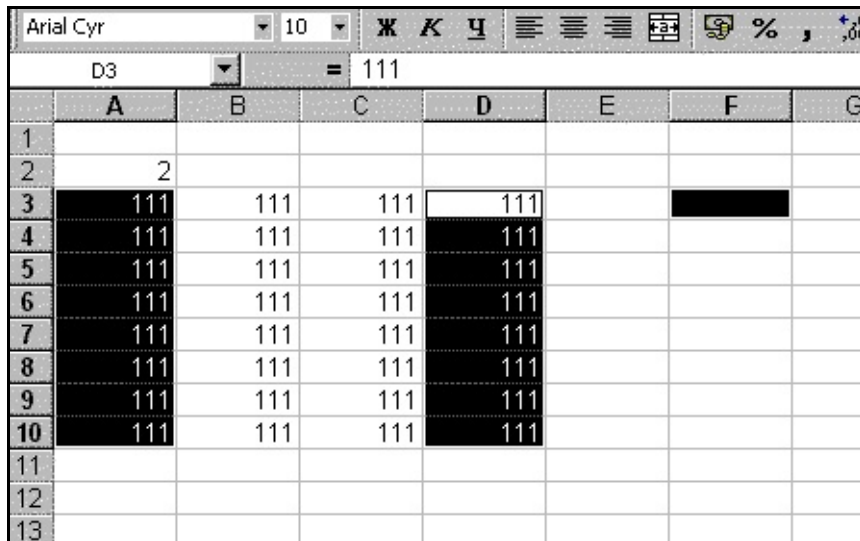
Автор [Каев Артем](#).

## Шаг 11 - Дальше о Range

Разговаривая о выделении ячеек с помощью **Range** мы с Вами должны знать, что возвращает этот метод множество. Это множество может состоять из одной или нескольких ячеек. А если множество, то информатика говорит о необходимости иметь возможность их объединения.

```
Sub Test()  
    With Application.Workbooks.Item("Т  
        Worksheets("Лист2").Activa  
        Dim HelloRange As Range  
        Set HelloRange = Range("D3  
        HelloRange.Select  
    End With  
End Sub
```

А вот результат работы этого кода.



	A	B	C	D	E	F	G
1							
2	2						
3	111	111	111	111			
4	111	111	111	111			
5	111	111	111	111			
6	111	111	111	111			
7	111	111	111	111			
8	111	111	111	111			
9	111	111	111	111			
10	111	111	111	111			
11							
12							
13							

Раз есть объединения должно быть и пересечение, раз уж идет разговор о теории множеств. Получить его можно вот так.

```

Sub Test()
    With Application.Workbooks.Item("Т
        Worksheets("Лист2").Activa
        Dim HelloRange As Range
        Set HelloRange = Range("A1
        HelloRange.Value = "Hello"
    End With
End Sub

```

Будет выделена всего одна ячейка. Как Вы видите из предыдущих примеров отличаются типом объявления - "D3:D10, A3:A10" это объединения, а "A1:A20 A8:D8" это пересечение. Используя пересечения и объединения можно строить область любого уровня сложности.

Получив объединение можно узнать количество ячеек.

```

Sub Test()
    With Application.Workbooks.Item("Т
        Worksheets("Лист2").Activa
        Dim HelloRange As Range
        Set HelloRange = Range("A1
        MsgBox (Str(HelloRange.Cou
    End With
End Sub

```

Да пора бы уже рассказать о этой **Str**. Эта функция переводит число в строку. Вот пример:

```

Sub Test()
    Dim x As Integer
    x = 10
    Dim s As String
    s = Str(x)
    MsgBox (s)

```

End Sub

Или так с типом **Double** она сама определит как переводить, простая и умная, как женщина моей мечты :-)

```
Sub Test()  
    Dim x As Double  
    x = 10.333333  
    Dim s As String  
    s = Str(x)  
    MsgBox (s)  
End Sub
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 12 - Обработка ошибок VBA

Программирование это как хождение по минному полю. Неизвестно где взорвешься. Наверно так. Вы слышаны о том, что **Windows** напичкан ошибками, о том что среда разработки любая при том - тоже. Мне попадались исследования на эту тему. Типа, что на каждые **1000** строк кода одна ошибка, у хорошего программиста естественно :-). В общем это закон такой. Все равно ошибешься где-нибудь. Проводя аналогию между женщиной и компьютером :-))) вообще понятно.

Для обработки ошибок в **VBA** и **VB** есть специальный оператор **On Error**. Его задача при возникновении ошибки передать управление в то место( процедура или кусок кода), в котором это ждут. Посмотрим пример:

```
Sub Test()  
    On Error GoTo Errors1  
    Dim x As Integer  
    Dim a As Integer  
    Dim c As Double  
    x = 20  
    a = 0  
    c = x / a  
    MsgBox (" Этого не должно быть")  
    GoTo Ends:  
Errors1:  
    MsgBox ("Ну ты блин Тикурила Даещь"  
Ends:  
End Sub
```

В данном примере при возникновении ошибки управление передается по метке **Errors1** и дальше выполняется код. Я понимаю, что прерывать функцию из-за ошибки не всегда надо. И не только я так думаю, создатели **VBA** тоже так считали, и поэтому есть оператор **Resume Next**. Этот оператор реализует небезызвестный принцип - Ни шагу назад. Выполнение

пойдет дальше, несмотря на ошибку.

```
Sub Test()  
    On Error GoTo Errors1  
    Dim x As Integer  
    Dim a As Integer  
    Dim c As Double  
    x = 20  
    a = 0  
    c = x / a  
    MsgBox ("Опаньки !!!")  
    GoTo Ends:  
Errors1:  
    MsgBox ("Ну ты блин Тикурила Даещь  
Resume Next  
Ends:  
End Sub
```

А вот, если Вы вообще не хотите ничего говорить по поводу ошибки, то можете поступить очень сурово. Вот так. Я рекомендую применять это для бухгалтерских расчетов. Ни кто и не догадается :-)))

```
Sub Test()  
    On Error Resume Next  
    Dim x As Integer  
    Dim a As Integer  
    Dim c As Double  
    x = 20  
    a = 0  
    c = x / a  
    x = 10  
    a = 3  
    c = 10 / 3  
    MsgBox ("Опаньки !!!")
```



End Sub

Над резюме можно немного поэкспериментировать, вот возможные описания:

```
Resume Next
Resume строка
Resume метка
Resume 0
```

Пример ниже будет упорно требовать, чтобы ввели число отличное от 0:

```
Sub Test()
    On Error GoTo Error1
    Dim x As Integer
    Dim a As Integer
    Dim c As Double
    x = 20
    a = Str(InputBox("введите число"))
    c = x / a
    x = 10
    MsgBox ("Опаньки !!!")
    GoTo Ends:
Error1:
    MsgBox ("думай о программировании,")
    a = Str(InputBox("введите число"))
    Resume 0
Ends:
End Sub
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 13 - Объект Err

Да, странное совпадение, 13 шаг и зловещий объект **Err**, от которого одни неприятности. Этот объект хранит информацию о последней ошибке в результате выполнения того, что вы запрограммировали. Ну давайте попробуем.

```
Sub Test()  
    On Error GoTo Error1  
    Sheets.Item(1000).Delete  
    GoTo Ends  
  
Error1:  
    MsgBox "Error detected"  
    MsgBox (Str(Err.Number))  
    MsgBox (Err.Source)  
    MsgBox (Err.Description)  
  
Ends:  
End Sub
```

Итак, **Number** - это номер ошибки **Source**, где она появилась, а **Description** описание. В данном случае Вам скажут о выходе за границу массива. Вот это здорово. Особенно при создании программ. Получить такое сообщение пользователю не очень приятно, а вот программисту :-)) даже думать не надо.

У объекта **Err** есть метод очистки **Clear**, он все очищает. Вот в этом случае Вы не получите никаких сообщений. После обработки ошибки неплохо применить этот метод. Так, ради профилактики.

```
Sub Test()  
    On Error GoTo Error1  
    Sheets.Item(1000).Delete  
    GoTo Ends  
  
Error1:
```

```
Err.Clear
MsgBox "Error detected"
MsgBox (Str(Err.Number))
MsgBox (Err.Source)
MsgBox (Err.Description)

Ends:
End Sub
```

Нельзя не сказать, что этот объект автоматически очистится после ..

```
Resume
Exit Sub(Function)
On Error
```

При отладке или специально в программе вы и сами можете сгенерировать ошибку методом **Raise**, только надо знать, что ошибки до **1000** зарезервированы **VBA**, а максимальный код **65535**. Любое правило подвержено изменениям и поэтому есть специальная константа, от которой вы можете сложением получать коды ошибок. Она называется **vbObjectError**.

```
Sub Test()
    On Error GoTo Error1
    Err.Raise 1001, "Test()", "Это я с,
Error1:
    MsgBox "Error detected"
    MsgBox (Str(Err.Number))
    MsgBox (Err.Source)
    MsgBox (Err.Description)

Ends:
End Sub
```

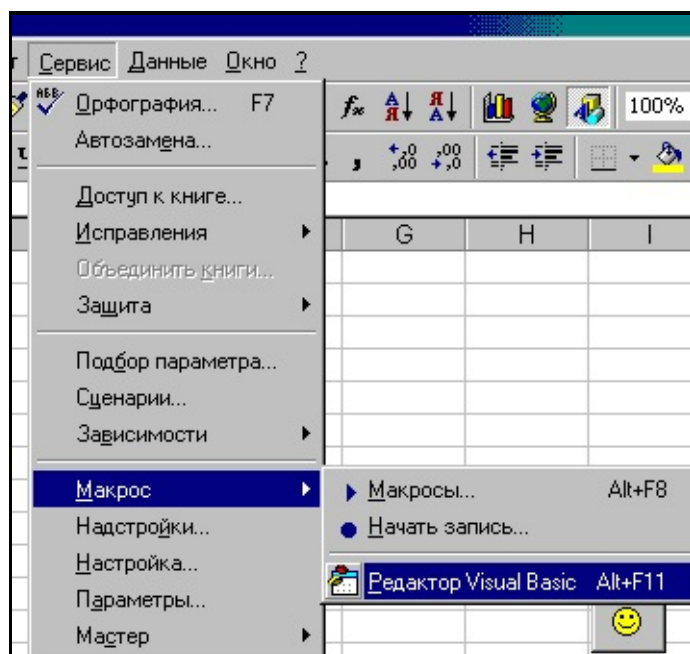
Автор [Каев Артем](#).

## Шаг 14 - События объектов

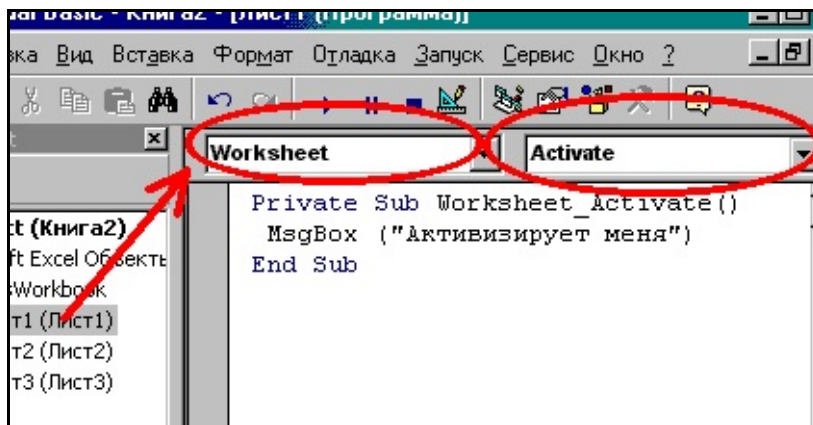
Обработать можно события следующих объектов **Excel**:

Application  
WorkBoor  
WorkSheet  
Chart

Функции обработки создаются автоматически. Открывайте **Excel** и любую книгу. Запускайте редактор **VBA**.



Появится редактор **VBA**. Нас интересует список объектов в окне **VBAProject**. Выберите **Лист1** и два раза шелкните по нему. Появится белое окно. Вам нужно выбрать объект и событие, смотрите как на рисунке ниже, да я чуть не забыл, код **VBA** еще нужен.



Пришло время испытаний. Переключитесь на **Лист2**, потом назад на **Лист1** должно появиться диалоговое окно о том, что активизирован лист. Это очень полезно. Например у Вас есть скрытый лист, пользователь открывает его и пробует смотреть, а Вы ему **format.com** за это :-). Хотя не смешно, Вам и восстанавливать.

Многие события имеют параметры. Вот как это.

```
Private Sub Worksheet_BeforeRightClick(ByVal Sh As Object, ByVal Target As Range)  
    .....  
End Sub
```

И еще события посылают не только по иерархии объектов вниз, но и вверх. Вот то же событие активации обрабатывается и на верхнем уровне.

```
Private Sub Workbook_SheetActivate(ByVal Sh As Worksheet)  
    MsgBox (Sh.Name)  
End Sub
```

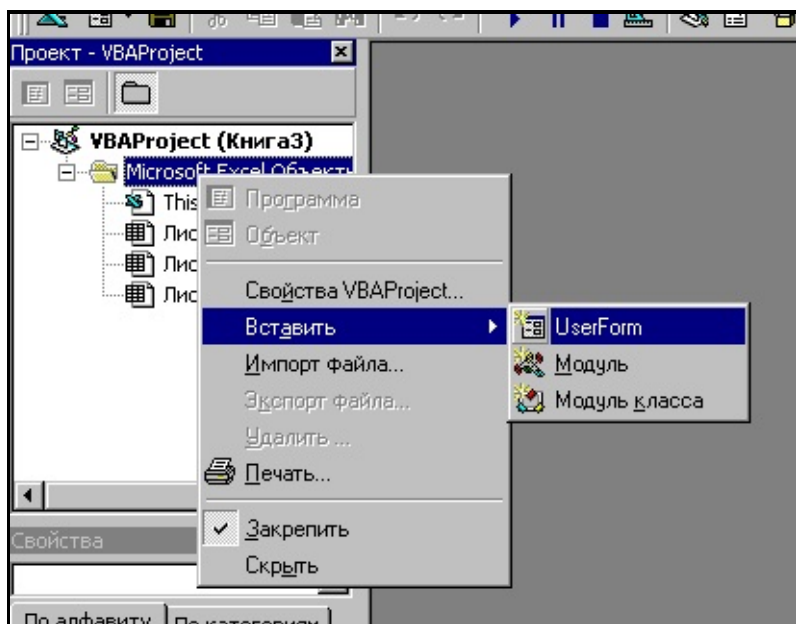
Попробуйте его создать и проверить.

---

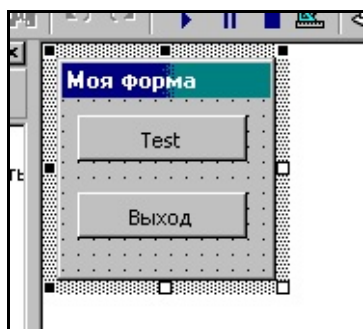
[Предыдущий Шаг](#) | [Следующий Шаг](#)  
Автор [Каев Артем](#).

## Шаг 15 - Пользовательские формы

Создавайте новую книгу. Запускайте редактор **VBA**, как в прошлый раз. Наводим курсор на **Microsoft Excel Объекты**, правую кнопку мыши, вставить **UserForm**. Да смотрите сами ниже. Чего это Я.



У Вас появится диалоговое окно (форма), панель инструментов, и окно свойств. В окне свойств нас интересует свойство **Caption**, которое позволяет нам изменить заголовок окна. Поменяйте его на осмысленное имя, например, первая форма :-). Разместите на форме кнопки. Выбираете инструмент кнопка. Не знаете какой он ? Подводите к каждому и задержите мышку, Вам подскажут. Нам надо две кнопки. Одна с именем "тест", а вторая с именем "выход". Имя у кнопок так же меняется в **Caption**. Все должно быть вот так.



Пора добавить к кнопкам код. Это просто. Двойной щелчок на кнопке и вы попадете в код вызываемый при нажатии кнопки. Давайте для кнопки **Test**.

```
Private Sub CommandButton1_Click()  
    MsgBox ("Test Button Press")  
End Sub
```

И для второй:

```
Private Sub CommandButton2_Click()  
    Unload Me  
End Sub
```

**Unload Me** выгружает форму из памяти. А вот теперь нам нужно создать макрос для загрузки. Создавайте макрос с именем **FormsRun** или другим. Код ниже.

```
Sub FormsRun()  
    UserForm1.Show  
End Sub
```

Вот теперь запустите макрос. Появится диалоговое окно. Нажмите кнопку **Test**, появится сообщение, нажмите кнопку **Выход**, окно диалога закроется.

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).



## Шаг 16 - Чтение и запись текстовых файлов

Наверно можно смело утверждать, что умение читать и записывать информацию в текстовый файл это основа импорта и экспорта :-)  
Практически любая серьезная программа хранящая информацию позволяет сохранить её в текстовом формате, какое бы он расширение не имел.  
Открываются файлы командой **Open**.

```
Sub Test()  
    Open "c:\1.txt" For Input As #1  
    Close #1  
End Sub
```

Команда **Open** может открывать для чтения **Input** и для записи **Output**.  
Цифра после **as** это идентификатор файла. На основании его производится чтение и запись файла.

Следующий пример демонстрирует запись и чтение файла

```
Sub Test()  
    Open "c:\1.txt" For Output As #1  
    Print #1, "Hello File"  
    Close #1  
  
    Open "c:\1.txt" For Input As #1  
    Dim s As String  
    Input #1, s  
    MsgBox s  
    Close #1  
End Sub
```

Как видите, для записи можно использовать **Print**, а для чтения **Input**

воспользовавшись идентификатором открытого файла. Естественно здесь свои тонкости работы. Вот, если Вы запишите такую строку:

```
.....  
Print #1, "Hello , File"  
.....
```

То оператор **Input #1** прочитает только **Hello** и все. Запятая воспринимается как разделитель. И это правильно. Есть форматы текстовых файлов когда числа разделены запятой. В коде ниже:

```
.....  
Input #1, s  
MsgBox s  
Input #1, s  
MsgBox s  
.....
```

Последовательно выведутся надписи **Hello** и **File**, но с этим можно бороться оператором **Line Input**.

```
Sub Test()  
    Open "c:\1.txt" For Output As #1  
    Print #1, "Hello , File"  
    Close #1  
  
    Open "c:\1.txt" For Input As #1  
    Dim s As String  
    Line Input #1, s  
    MsgBox s  
    Close #1  
End Sub
```

Этот код прочитает строку целиком. Следом возникает важный вопрос, а

как узнать конец файла ? Для этого есть функция **EOF(идентификатор)**, которая позволяет вам определить конец файла.

```
Sub Test()  
Open "c:\1.txt" For Output As #1  
    Print #1, "Hello , File"  
Close #1  
Open "c:\1.txt" For Input As #1  
    Dim s As String  
    While Not EOF(1)  
        Input #1, s  
        MsgBox s  
    Wend  
Close #1  
End Sub
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 17 - Win32 API и VBA

На данный момент использование **Win32 API** является стандартом для любой среды или языка программирования, это и понятно, как иначе писать программы для **Windows** ? Вместе с тем пользоваться этим же **API** надо осторожно, реализации в версиях **Windows** отличаются вплоть до присутствия некоторых функций. Для того, чтобы использовать функции **Win 32 API** их необходимо объявить, используя **Declare**.

В общей области (в описании) надо объявить функцию. Сделать это можно поднявшись на самую верхнюю строчку окна редактирования макроса и ввести описание.

```
Declare Function GetWindowsDirectory Lib "
```

Вот тут-то Вы и должны быть поражены. Говорят **VBA** это для ..... Так вот. Использовать подобную функцию на **VC++** намного проще. Во-первых, Вам наверно всё равно, где она находится :-))) в **kernel**, **user** или **gdi**, и вам вообще-то и не надо знать её имя в виде **GetWindowsDirectoryA**, а если вы пользуетесь каркасной библиотекой типа **MFC**, то часто получаете упрощенный вид функции типа **AfxMessageBox**. Вот и думай теперь чего проще :-))

Давайте на **Declare** посмотрим повнимательнее. У него два синтаксиса для функции или процедуры.

```
REM то что в скобках необязательно  
[Public или Private] Declare Sub имя_проц,  
[Public или Private] Declare Function имя_
```

Вот так надо знать где находится и псевдоним, если нужно и все параметры. Вот он язык для домохозяек :-)

А теперь применение. Вот тут все стало опять просто.

```
Sub Test()  
    Dim buffer As String  
    Dim lens As Long  
    buffer = String(256, 0)  
    lens = GetWindowsDirectory(buffer,  
    buffer = Left(buffer, lens)  
    MsgBox (buffer)  
End Sub
```

Объявляю переменные, **buffer = String(256, 0)** - заполняю строку нулями имитируя строку символов **char**. Зачем ? Ну есть подозрение, что **Windows** написан на **C** или **ASM**, даже без ++ и поэтому другого он не понимает :-), не на бейсике точно. Вызываем функцию, передавая параметры. Полученную строку обрезаем функцией **Left**.

Наверно лучше создавать описания в отдельном модуле и просто его экспортировать в проект, дабы не мучаться. И, наверно, есть уже готовые модули. Но этот метод позволяет Вам подключить любую динамическую библиотеку. Посмотрите в разделе **MFC** шаг за шагом : ["Шаг 46 - Dll для Excel"](#).

---

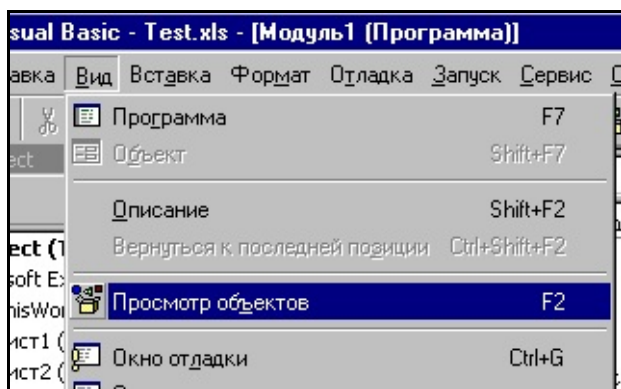
[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

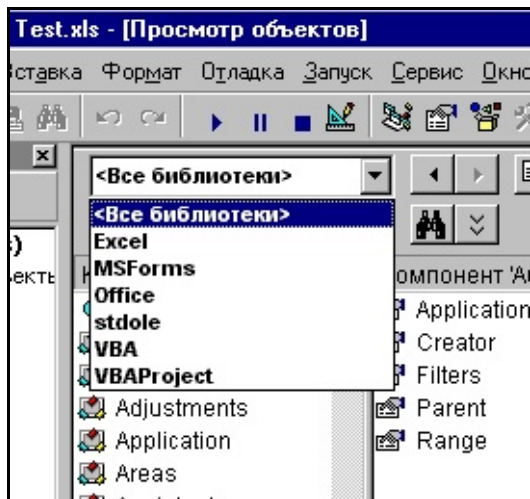
## Шаг 18 - Просмотр объектов

Весь **Office 97** можно рассматривать как набор объектов. Кроме того операционная система предоставляет дополнительные объекты. Каждый объект имеет свои свойства. Объекты операционной системой предоставляются с использованием технологии **OLE** и интерфейсов следующих поколений на базе него.

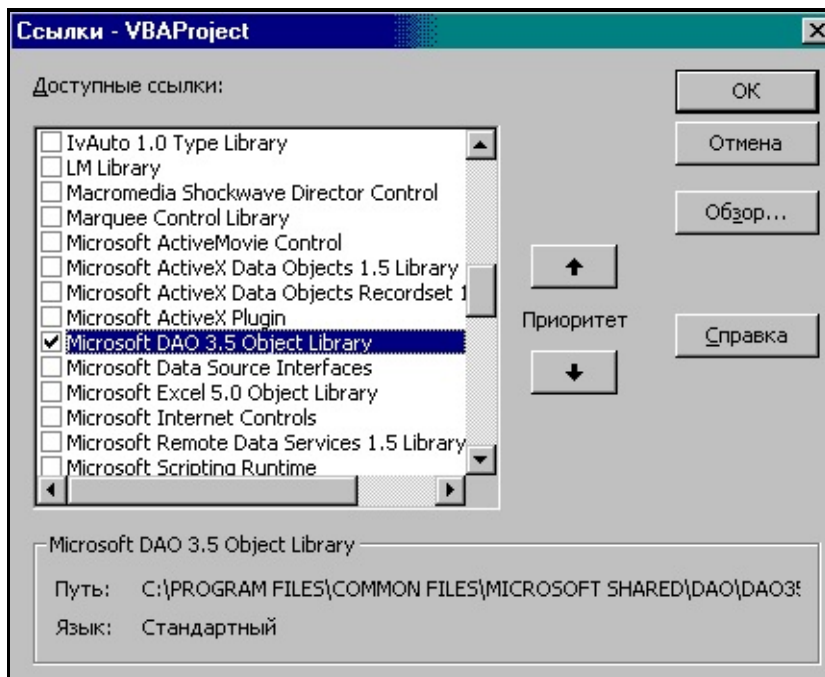
Редактор **VBA** позволяет вам просмотреть доступные Вам объекты и их свойства. Для этого Вам необходимо запустить **Просмотр объектов** из меню **Вид (F2)**.



Появится окно. Оно разделено на 3 части. Вверху библиотека, справа объекты библиотек, а слева свойства. По умолчанию у Вас подключаются библиотеки необходимые по мнению создателей **Office**. В верхнем окне у Вас должно быть написано **Все библиотеки**. Давайте посмотрим какие присутствуют.



Как вы видите количество библиотек ограничено. Вот тут Вы должны возмутиться, а где библиотеки **ACCESS** или **DAO**. И вообще мало .... Не мало. Просто их необходимо подключить. Точнее надо. Если вы будете работать с базами данных или хотите расширить возможности среды, то их нужно подключать. Делается это из меню **Сервис-Ссылки**. Выберите этот пункт меню.



Подключаем **Microsoft DAO 3.5 Object Library**. Без этой библиотеки Вы не сможете работать с базами данных на основе **DAO**. Только библиотека должна быть зарегистрирована в системе, иначе в списке её не будет. Что

делать тогда ? В окне подключения библиотек есть кнопка **Обзор**, которая позволяет Вам подключить их используя, например, **TLB** файлы.

Что же произойдет после подключения? В списке объектов появятся **DAO** и все связанные с ним свойства.

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).



## Шаг 19 - Информация о типе переменной

А зачем, вы спросите, иметь информацию о типе переменной в ходе работы программы ? Ведь это делает программист. Опаньки :-). В **VBA** есть тип переменной **Variant**, который может быть любого типа за исключением пользовательского. Не верите ? Смотрите код:

```
Sub Test()  
    Dim string_var As String  
    Dim int_var As Integer  
    Dim test_variant As Variant  
    string_var = "Hello Variant"  
    int_var = 123  
    test_variant = string_var  
    test_variant = int_var  
End Sub
```

Как видите, и **Variant** можно передавать в процедуры, поэтому определение типа нужно, конечно если подобными вещами вы будете пользоваться. Для определения кода есть функция **TypeName (...)**, которая вернет строку с именем переменной. Вот так, например, можно её использовать:

```
Sub Test()  
    Dim string_var As String  
    Dim int_var As Integer  
    Dim test_variant As Variant  
    string_var = "Hello Variant"  
    int_var = 123  
    test_variant = string_var  
    MsgBox (TypeName(test_variant))  
    test_variant = int_var  
    MsgBox (TypeName(test_variant))
```

```
End Sub
```

Кроме этого есть ряд вспомогательных функций позволяющих получить информацию о переменных. **IsArray** позволяет проверить является ли переменная массивом.

```
Sub Test()  
    Dim arr_var(10) As String  
    If IsArray(arr_var) Then  
        MsgBox ("Массив")  
    End Sub
```

**IsEmpty** проверка инициализации (наличия) переменной. Запустите код ниже, а потом раскомментируйте строку.

```
Sub Test()  
    ' Dim arr_var As String  
    If IsEmpty(arr_var) Then MsgBox ("")  
End Sub
```

**IsDate** проверяет можно ли преобразовать переменную к типу даты. Ниже надпись **YES** появится один раз.

```
Sub Test()  
    Dim arr_var As String  
    arr_var = "01.01.1998"  
    If IsDate(arr_var) Then  
        MsgBox ("YES")  
    arr_var = "41.01.1998"  
    If IsDate(arr_var) Then  
        MsgBox ("YES")  
    End Sub
```

Так же проверяется можно ли перевести в число **IsNumeric**:

```
Sub Test()  
    Dim arr_var As String  
    arr_var = "not numeric"  
    If IsNumeric(arr_var) Then MsgBox  
    arr_var = "1998"  
    If IsNumeric(arr_var) Then MsgBox  
End Sub
```

Есть еще ряд подобных функций:

- **IsObject** - проверка, что переменная объект
- **IsNull(выражение)** - проверка на пустое значение
- **IsError(выражение)** - проверка выражения, представляет ли оно значение ошибки

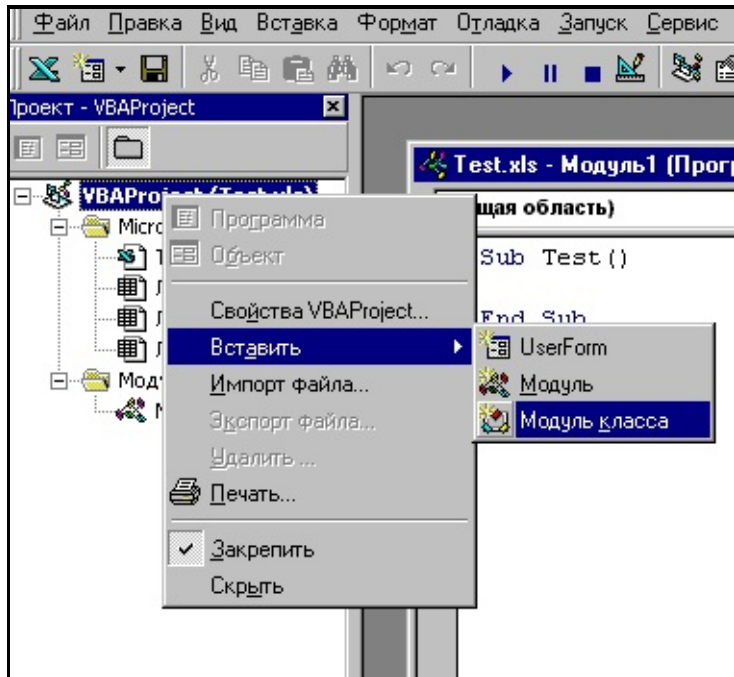
---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 20 - Пользовательские классы

В **VBA** есть свои классы, но можно создавать и самим. Для этого в проект необходимо добавить модуль класса. Это можно сделать щелкнув правой кнопкой мыши на проекте и выбрав пункт меню **вставить** -> **модуль класса**.



В результате у Вас появится окно для кода класса, и в окне просмотра проекта появится значок класса. Вероятнее всего с именем **Класс1**. Объявим переменные для внутреннего использования. **Private** говорит о том, что использоваться они будут только внутри класса.

```
Private NamePipe As String  
Private DatePipe As String
```

Теперь создадим функцию **GetPipeName**. Пишите ниже:

```
Public Sub GetPipeName()  
    NamePipe = InputBox("Enter Name -"  
End Sub
```

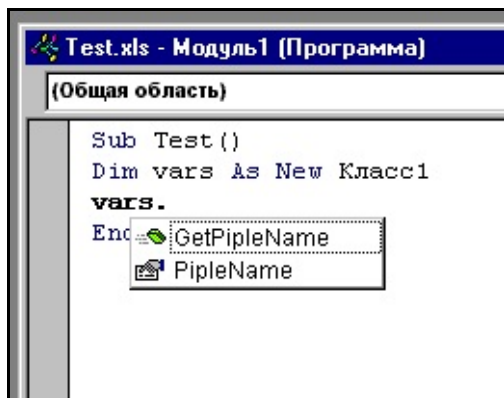
Теперь свойства для получения имени. Пишите ниже:

```
Property Get PipleName() As String
    PipleName = NamePiple
End Property
```

И для установки тоже. Пишите ниже:

```
Property Let PipleName(s As String)
    NamePiple = s
End Property
```

Закрывайте редактор и открывайте любой макрос для редактирования, если его нет создайте. Начинайте вводить код, как на рисунке ниже. И о чудо !!! Наш класс имеет те же возможности, что и встроенный класс **VBA**, он показывает свойства.



Настало время испытать его в действии:

```
Sub Test()  
    Dim vars As New Класс1  
    vars.GetPipleName  
    MsgBox vars.PipleName  
    vars.PipleName = "VBA"  
    MsgBox vars.PipleName  
End Sub
```

По удобству и простоте это круче **C++** и **MFC** и так далее. Кроме того класс легко сохранить для дальнейшего использования. Вообще класс. Просто оцените эту возможность даже если вы читаете просто так.

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

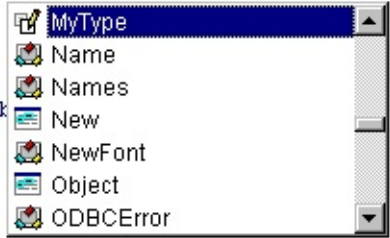
## Шаг 21 - Пользовательские типы

В общей части Вы должны описать структуру и объявить переменную типа этой структуры. Смотрите картинку.

```
Private Type MyType
    x As Integer
    y As Integer
    c As Variant
End Type
Sub Test ()

dim a as My

End Sub
```



Итак мы создадим тип с элементами **x,y**, типа **Integer** и **c** с типом **Variant**. Ох уж этот **Variant**, он позволяет нам в структуру помещать все, что угодно. Хоть массив. Мы то сделаем. Но разве Вас не впечатляет простота и мощь. Ведь так можно создать структуры любой сложности очень просто. Нет **OLE**, нет указателей. После **C++** это просто сказка.

' Описание

```
Private Type MyType
    x As Integer
    y As Integer
    c As Variant
End Type
```

```
Sub Test()
```

' -----

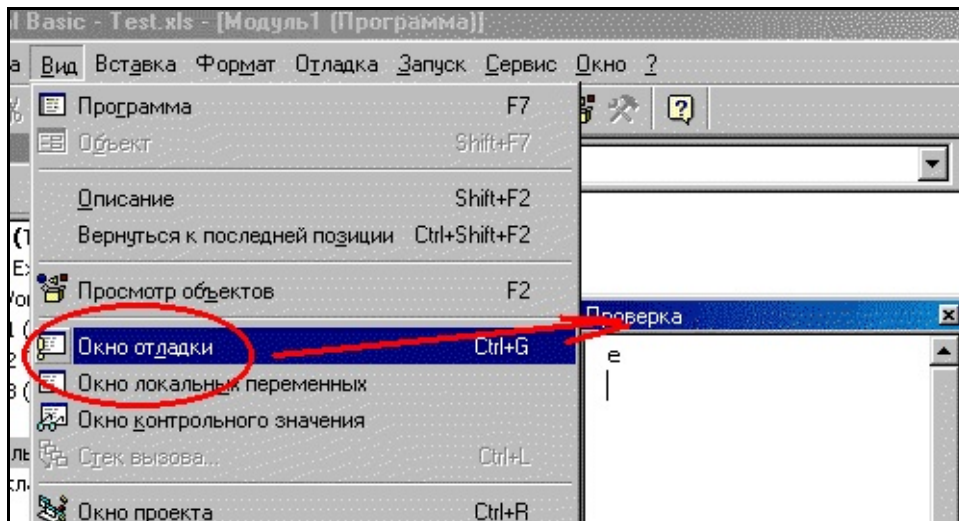
' Область функции

```
Sub Test()  
    Dim a(2) As MyType  
    Dim arrays(2) As String  
    arrays(1) = "Hello"  
    arrays(2) = "Cool"  
    a(1).x = 1  
    a(1).y = 2  
    a(1).c = arrays  
    a(2).x = 10  
    a(2).y = 20  
    arrays(1) = "Hello 2"  
    arrays(2) = "Cool !!!"  
    a(2).c = arrays  
    Debug.Print "a1"  
    Debug.Print a(1).c(1)  
    Debug.Print a(1).c(2)  
    Debug.Print "a2"  
    Debug.Print a(2).c(1)  
    Debug.Print a(2).c(2)  
End Sub
```

Во всем этом есть и подводный камень. Создавать структуры можно только на уровне модуля. Не зря она у нас **Private**. Но можно создавать классы, которые умеют с ними работать :-). Это несколько радует.

Для просмотра результата Вам необходимо открыть окно отладки. Это можно сделать в меню **Вид - Окно отладки (CTRL-G)**.





Вот такой результат вы должны увидеть:

```
a1  
Hello  
Cool  
a2  
Hello 2  
Cool !!!
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)  
Автор [Каев Артем](#).

## Шаг 22 - For Each

Этот цикл придуман для того, чтобы облегчить действия над массивами и наборами. Он позволяет произвести однотипные операции над всем массивом. Описание его такое:

**For Each** переменная **In** массив

Действия

**Next** переменная

Переменная должна иметь тип **Variant** или **Object**, в котором можно хранить практически всё. Пример? Пожалуйста.

```
Sub Test()  
    Dim arrays(1) As String  
    arrays(0) = "Hello"  
    arrays(1) = "Each :-)"  
    Dim vari As Variant  
    For Each vari In arrays  
        MsgBox (vari + " - Steps")  
    Next vari  
End Sub
```

**For Each** очень удобен для работы с коллекциями. Вот так можно пробежаться по открытым кигам.

```
Sub Test()  
    Dim vars As Variant  
    For Each vars In Workbooks  
        MsgBox (vars.Name)  
    Next vars
```

End Sub

Или по листам книги:

```
Sub Test()  
    Dim vars As Variant  
    For Each vars In Workbooks.Item("Т  
        MsgBox (vars.Name)  
    Next vars  
End Sub
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 23 - Работа с каталогами

Первым делом определим откуда запущено приложение. У объекта **Application** есть свойства **path**, которое и позволит нам получить информацию.

```
Sub Test()  
    MsgBox (Application.Path)  
End Sub
```

Команды создания и удаления каталогов очень похожи на **DOS** аналоги. Это **MkDir** и **Rmdir**. Ниже создаем каталог на диске **C**.

```
Sub Test()  
    MkDir ("c:\test")  
End Sub
```

И удаляем.

```
Sub Test()  
    Rmdir ("c:\test")  
End Sub
```

А вот теперь важный вопрос. Помещается ли удаленный каталог в корзину. Нет не помещается. Это очень возмутительно. Почему ????. Ведь программа создана для **Windows** с использованием среды разработки от **Microsoft** и такое возмутительное безобразие. Ладно **Linux** им судья. Для того, чтобы убедиться в этом запустите следующий код и загляните в корзину.

```
Sub Test()  
    MkDir ("c:\test")  
    Rmdir ("c:\test")  
End Sub
```

Для получения текущего каталога есть функция **CurDir**.

```
Sub Test()  
    MsgBox (CurDir)  
End Sub
```

Для того, чтобы сменить каталог тоже есть функция - **chdir**:

```
Sub Test()  
    ChDir ("c:\windows")  
    MsgBox (CurDir)  
End Sub
```

Команда **dir** позволяет просмотреть все файлы в каталоге. Только использование её несколько специфично. Сначала Вы вызываете **dir** с параметрами и получаете первое имя файла, в дальнейшем можно вызвать её без параметров и получить следующее имя и так до тех пор, пока не вернется пустое имя файла.

```
Sub Test()  
    Dim s As String  
    s = Dir("c:\windows\inf\*.*.*)  
    Debug.Print s  
    Do While s  
  
        s = Dir  
        Debug.Print s  
    Loop  
End Sub
```

Результат работы в Окне отладки (**Ctrl-G**).

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 24 - Использование Automation

Использование элементов **ActiveX** на базе модели **COM** - компонентная модель объектов, позволяет создавать сложные составные документы, то есть там могут находиться материалы из разных программ - **Excel**, **Access**, **PowerPoint** и так далее. Кроме этого есть возможность пользоваться другим приложением для решения задач. Например **Excel** может использовать **Access** для хранения данных или наоборот **Access** может использовать **Excel** для расчетов. Вобщем это можно назвать построением пользовательских приложений на базе готовых программ.

Понятие, которое используется в основе всех интегрированных систем является служба. **MS OFFICE** обеспечивает все необходимые службы для создания офисных приложений:

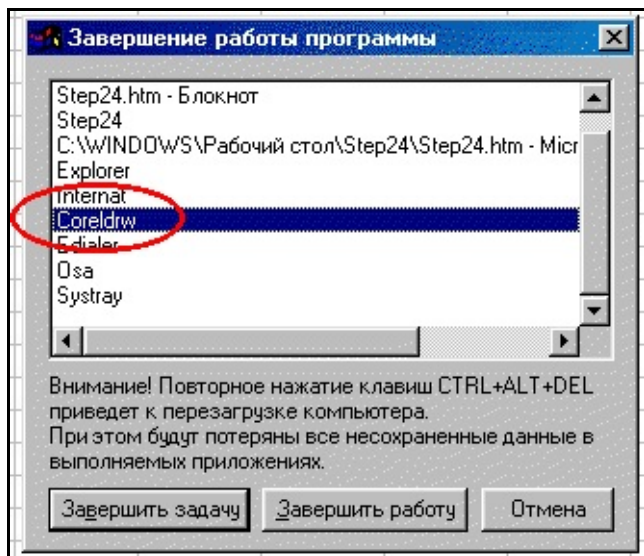
- **ACCESS** - База данных
- **EXCEL** - Расчеты
- **WORD** - Текстовый редактор
- **PowerPoint** - Презентационная графика
- **Office Binder** - Интеграция документов
- **Outlook** - Служба управления документами
- **Internet Exploler** - Работа с интернет

Объект с вашим приложением можно связать используя позднее и ранее связывание. Позднее связывание происходит на этапе выполнения кода и для него используется понятие **Object**. Ниже будет приведен код для программы **Corel Draw** и использование её в качестве объекта для **Automation**.

```
Sub Test()  
    Dim objCorel As Object  
    Set objCorel = CreateObject("Corel
```

```
MsgBox ("press")  
End Sub
```

В момент когда на экране появится сообщение **PRESS** нажмите **Ctrl-Alt-Delete** для просмотра активных объектов. Вот смотрите ниже.



Для позднего связывания используется меню **Сервис - Ссылки**, в предыдущих шагах мы об этом пункте меню упоминали. Вот пример для **Excel**.

```
Sub Test()  
    Dim objExcel As Excel.Application  
    Set objExcel = CreateObject("Excel")  
End Sub
```

Ну и напоследок как можно использовать объект **Word** из **Excel**:

```
Sub Test()  
    Dim objWord As Word.Application  
    Set objWord = CreateObject("Word.Application")  
    MsgBox (objWord.Caption)  
    MsgBox (objWord.UserName)  
End Sub
```



---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 25 - О функции SendKeys

Эта функция позволяет имитировать ввод с клавиатуры в Окно вот её описание:

`SendKeys` строка, [режим ожидания]

Этот макрос прокрутит таблицу на страницу вниз.

```
Sub Test()  
    SendKeys ("{PGDN}")  
End Sub
```

Режим ожидания это как будет произведен возврат. Если **TRUE** возврат в процедуру будет только после обработки кодов. Обработка может быть длительной, если у Вас есть обработчики событий. **FALSE** вернет сразу ничего не ожидая.

Вы обратили внимание на фигурные скобки. В них указываются команды и символы:

```
+  
^  
%  
~  
(  
)  
DEL {DEL}  
INS {INS}
```

и так далее :-) догадаетесь?

```
{BS} {BREAK} {CAPSLOCK} {ENTER} {DOWN} {PGI
```

Это не все, но направление понятно.

Функция ниже переведет указатель на страницу ниже, введет **123** и даже **ENTER** нажмет :-)

```
Sub Test()  
    SendKeys ("{PGDN}")  
    SendKeys ("123{ENTER}")  
End Sub
```

Вот так можно вызвать функциональную клавишу:

```
Sub Test()  
    SendKeys ("{F1}")  
End Sub
```

Когда экспериментируете запускайте макрос из активной рабочей книги.

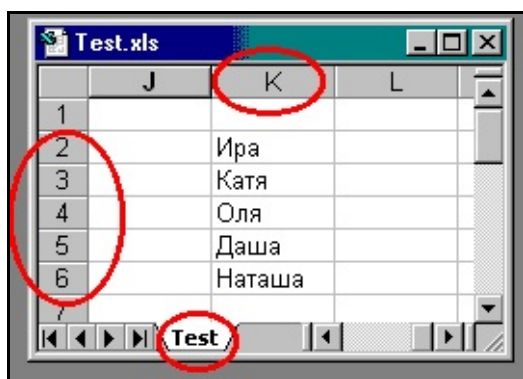
---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

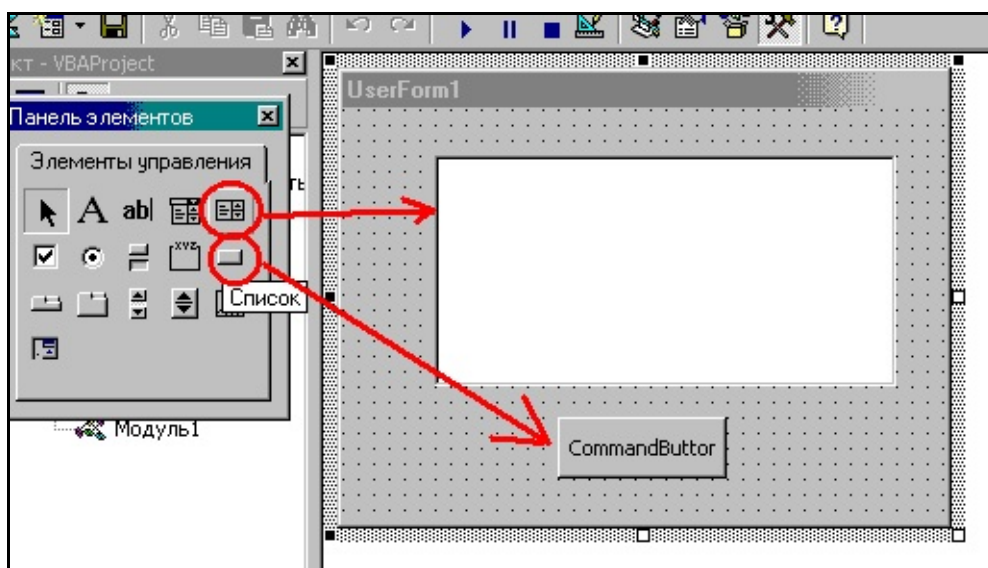
## Шаг 26 - Заполнение списка на форме из таблицы

Итак, задача простая. У нас есть в таблице список девушек и мы ходим создать макрос, который будет зачитывать этот список и выводить в диалоговом окне. Вот под эту таблицу создавался макрос.



Создайте точно такую таблицу, если не трудно :-)

Теперь переходим в редактор **Visual Basic** и создаем форму. На эту форму надо поместить два элемента управления: кнопку и список. Вот она такая.



Щелкайте два раза по кнопке и вы попадете в редактирование события нажатия. Введите код:

```
Private Sub CommandButton1_Click()  
    Unload Me  
End Sub
```

Что означает "уничтож меня" :-). То есть форму. Это **me** похожа на **this** в C++ и идентифицирует объект, в котором производятся события. Даже спрашивать не надо, где я нахожусь. **Unload** и всё.

Заполнять список мы будем при активизации формы. Поэтому нам необходимо обработать событие инициализации. Щелкните по форме два раза и выберите из меню событий (справа) **Activate**. И код.

```
Private Sub UserForm_Activate()  
    ' Активизируем нужный лист  
    Worksheets.Item("Test").Activate  
    ' Выбиделяем диапазон  
    Dim девушки As Range  
    ' Объект выделения  
    Set девушки = Range("K2:K6")  
    ' Выделяем  
    Dim vars As Variant  
    ' Пойдем по девчатам :-)  
    For Each vars In девушки  
        ' Добавляем в список  
        UserForm1.ListBox1.AddItem  
    Next vars  
End Sub
```

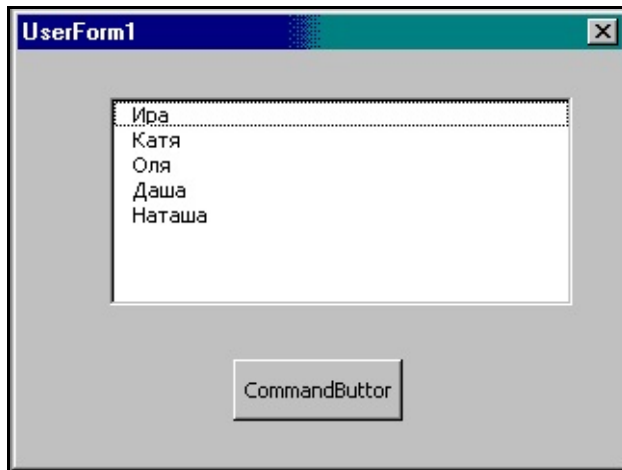
Код прокомментирован и наверно понятен. Использование русских слов для переменных это не ошибка. Наконец это делать можно. Здесь в **VBA** можно использовать русские буквы для имен переменных.

И макрос для запуска формы:

```
Sub Test()
```

```
UserForm1.Show  
End Sub
```

В макросе написано - "форма покажись". Ну вот. Запускайте. Вот как это получится:



---

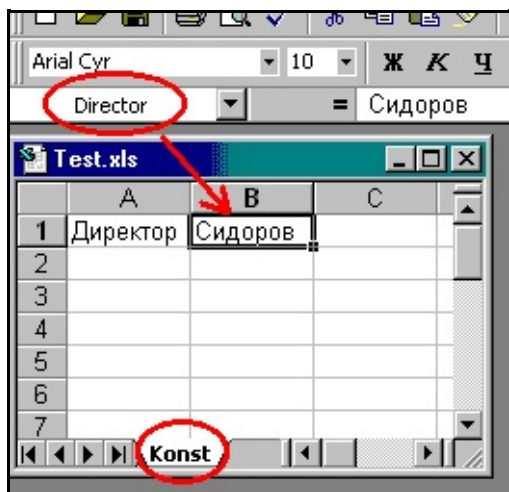
[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

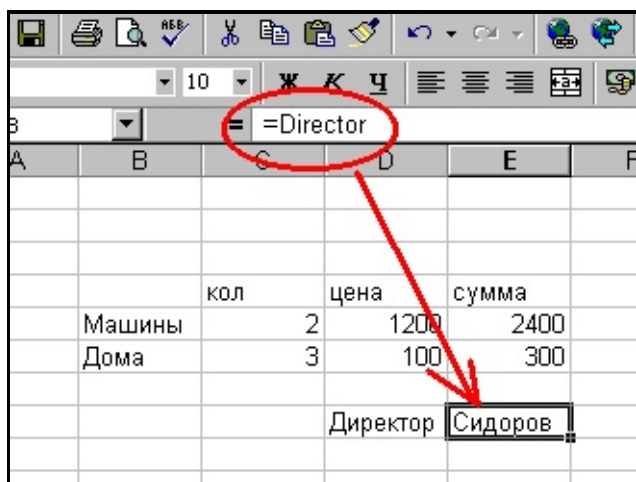
## Шаг 27 - Обмен данными между формой и таблицей

Задача. Я хочу сделать ряд справок и страницу с константами. Одной из констант будет фамилия директора, которая используется в справках. При изменении этой константы фамилия в справке должна автоматически меняться. И я хочу менять фамилию из формы.

Создаем лист Константы и на нем ячейке даем имя.

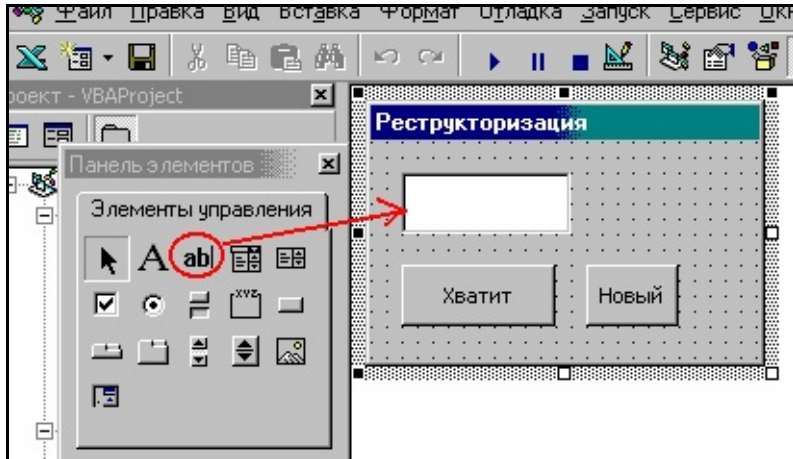


И любое количество листов со справками. Ссылаясь на ячейку с фамилией.



Как вы понимаете, сколько я листов не создам, воспользовавшись ссылкой

на ячейку =**director**, стоит мне изменить данные в ячейки с фамилией директора она везде поменяется. Это само нормально. Вот только менять я хочу из формы, например, чтобы с константами спрятать лист подальше от пользователя. Ну давайте создавать форму. Идем в редактор **VBA**:



При запуске формы мы должны прочитать данные с листа:

```
Private Sub UserForm_Activate()  
    Worksheets.Item("Konst").Activate  
    UserForm1.TextBox1.Text = Range("D  
End Sub
```

При нажатии на кнопку "**Новый**" заменить данные на листе константы (автоматически поменяются на справках):

```
Private Sub CommandButton2_Click()  
    Range("Director").Value = UserForm  
End Sub
```

По нажатию на "**Хватит**" закрыть форму:

```
Private Sub CommandButton1_Click()  
    Unload Me  
End Sub
```



Как видите использование констант в тех случаях, где это разумно на отдельном листе позволяет просто создать форму. Потом форму можно скрыть и вызвать из меню для замены значений.

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 28 - Работа с Датами

Для работы с датой в **VBA** предусмотрен специальный тип **Date**. Этот тип занимает 8 байт. Оно вам надо ? Это так для информации :-). Пробуем.

```
Sub Test()  
    Dim MyDate As Date  
    MsgBox (Str(Year(MyDate)))  
End Sub
```

У меня выдает 1899 год. Это говорит, что при создании этой переменной она не инициализируется текущей датой. Это плохо. Поместить Дату и время можно из строки воспользовавшись функциями **DateValue** и **TimeValue**.

```
Sub Test()  
    Dim MyDate As Date  
    MyDate = DateValue("1/1/96")  
    Debug.Print Year(MyDate)  
End Sub
```

Так же и со временем:

```
Sub Test()  
    Dim MyDate As Date  
    MyDate = TimeValue("10:10:12")  
    MsgBox Str(Minute(MyDate))  
End Sub
```

Только одновременно хранить и время и дату так не удастся, вот этот код приведет к очень интересному результату.

```
Sub Test()
```

```

        Dim MyDate As Date
        MyDate = DateValue("6/1/72")
        MsgBox Str(Year(MyDate))
        MyDate = TimeValue("10:10:12")
        MsgBox Str(Minute(MyDate))
        MsgBox Str(Year(MyDate))
    End Sub

```

Если вы хотите хранить вместе и дату и время, то поступите так:

```

Sub Test()
    Dim MyDate As Date
    MyDate = DateValue("6/1/72") + TimeValue("10:10:12")
    MsgBox Str(Minute(MyDate))
    MsgBox Str(Year(MyDate))
End Sub

```

Чтобы извлекать части даты и часов используйте такие функции:

```

Month(переменная типа Date)
Day(переменная типа Date)
Year(переменная типа Date)
Hour(переменная типа Date)
Minute(переменная типа Date)
Second(переменная типа Date)
WeekDay(переменная типа Date)

```

**WeekDay** - это день недели, если Вам это нужно, то вы можете написать что-то типа этого.

```

Sub Test()
    Dim MyDate As Date
    MyDate = DateValue("9/1/72")
    If (WeekDay(MyDate) = vbSunday) Then

```

MsgBox ("Sunday")  
End Sub

**vbSunday** это константа, есть еще **vbMonday**, ну дальше понятно.

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 29 - Использование With

Оператор **With** используется для явного указания объекта, к свойствам которого мы хотим получить доступ. Вот так это выглядит в глобальном плане.

```
With объект
    операции с объектом
End With
```

Давайте рассмотрим пример. Ниже реализованы два сообщения, которые выводят имя и статус видимости объектов:

```
Sub Test()
    MsgBox (Application.Worksheets.Item(1).Name)
    MsgBox (Str(Application.Worksheets.Item(1).Visible))
End Sub
```

Используя **With** это можно сделать так:

```
Sub Test()
    With Application
        With .Worksheets
            MsgBox (.Item(1).Name)
            MsgBox (Str(.Item(1).Visible))
        End With
    End With
End Sub
```

Используя **With** можно получить доступ и к пользовательским структурам.

```
' ----- Описание -----
Type Family
```

```
        Name_I As String
        Name_Cat1 As String
        Name_Cat2 As String
End Type

'----- Код -----

Sub Test()
Dim fam As Family

With fam
    .Name_I = "Pety"
    .Name_Cat1 = "Vasi"
    .Name_Cat2 = "Fisa"
    MsgBox (.Name_I)
End With

End Sub
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)  
Автор [Каев Артем](#).

## Шаг 30 - Рекурсия в VBA

При программировании многие задачи решаются на основе рекурсии. Т.е. есть ряд задач, которые вообще без рекурсии не решаются. Это задачи имитации человеческого интеллекта на основе перебора вариантов. Без рекурсии есть возможность решить подобные задачи только для частных случаев. Понятие рекурсии довольно молодое. Вот справка:

1958 год. В руководстве по программировании появилось понятие рекурсивности. Рутисхауз

Рекурсивная процедура - это процедура вызывающая сама себя. Классический пример подсчет факториала. Мы то его и реализуем:

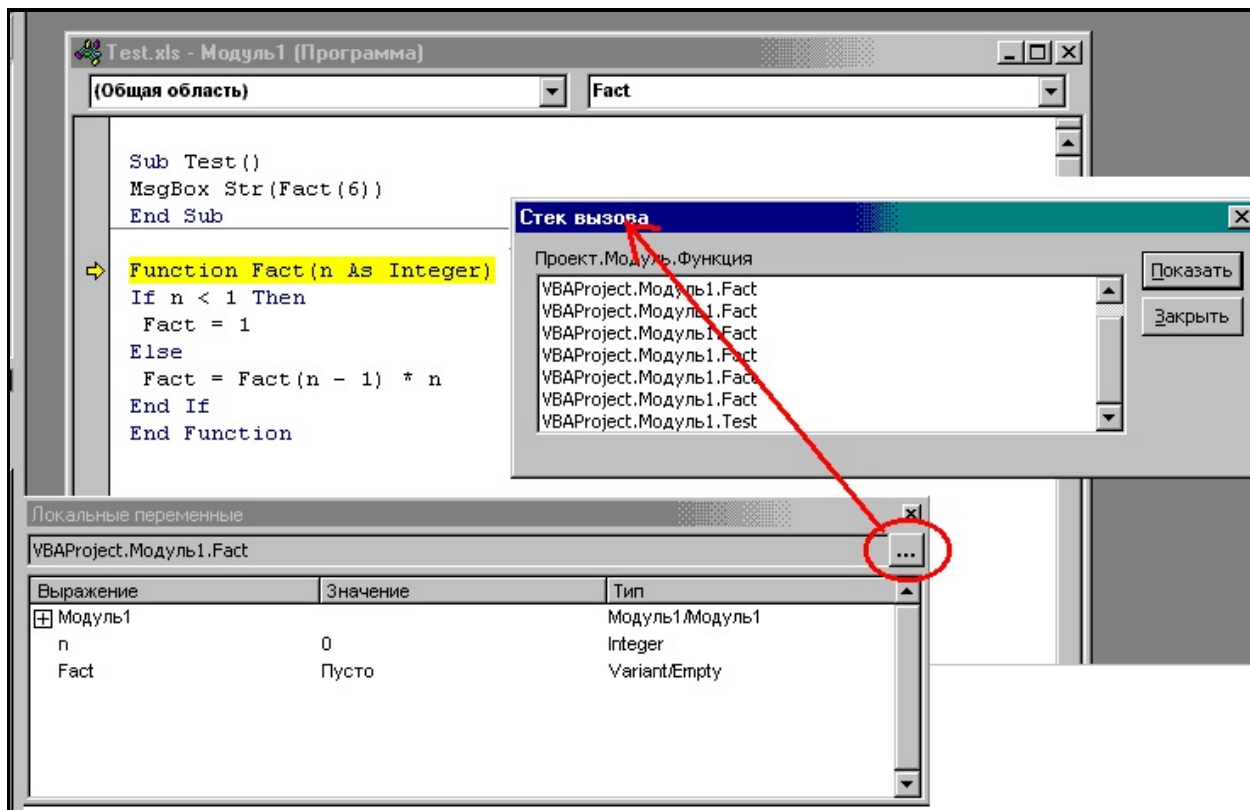
```
Sub Test()  
    MsgBox Str(Fact(3))  
End Sub  
  
Function Fact(n As Integer)  
    If n < 1 Then  
        Fact = 1  
    Else  
        Fact = Fact(n - 1) * n  
    End If  
End Function
```

Всё это хорошо, только для рекурсивных функций используется стековая память, которая имеет предел :-). В этой памяти размещаются и аргументы. Если их много или они большие по памяти хранения - финиш настанет еще быстрее. С рекурсивными функциями связано еще и время выполнения. То же в плохую сторону.

В **Excel** нет рекурсивных объектов. Листы, книги, ячейки не рекурсивные. Но вот данные :-) им всё ни почем. Вы можете создавать используя **Type**

структуры и создавать деревья. Для обработки их удобно использовать рекурсию.

Что происходит при вызове рекурсивных процедур можно увидеть выполняя программу по шагам (F8) и просматривая окно локальных переменных или сразу стек вызовов из меню Вид.



Я тут ставил эксперимент и оказалось, что факториал числа больше 100 можно рассчитать, но вот 200 уже нельзя. Переполнение говорит. Вот так.

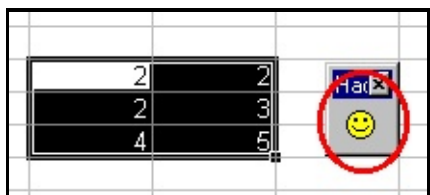
[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).



## Шаг 31 - Работаем с выделенным диапазоном

Наша задача научиться обрабатывать выделенный диапазон. Я надеюсь, что кнопка до сих пор связана у Вас с макросом. Как ниже. То есть пользователь выделяет диапазон, а по нажатию на кнопку над ним производится работа. Например умножения всех чисел на два.



Попробуем получить информацию о выделенном диапазоне:

```
Sub Test()  
    ' объявим переменную типа Range  
    Dim cur_range As Range  
    ' активный расчетный лист  
    With ActiveSheet  
        ' объект Range включает вы  
        Set cur_range = Selection  
        ' активизируем Range  
        cur_range.Activate  
        ' Адрес и количество строк  
        Debug.Print cur_range.Address  
        Debug.Print cur_range.Columns  
        Debug.Print cur_range.Rows  
    End With  
End Sub
```

А вот и код. Ниже написана функция, которая значения в ячейках умножит на 2. Будь то одна ячейка или диапазон ячеек.

```

Sub Test()
    Dim cur_range As Range
    With ActiveSheet
        Set cur_range = Selection
        cur_range.Activate
        For x = 1 To cur_range.Rows.Count
            For y = 1 To cur_range.Columns.Count
                ' значению ячейки
                cur_range(
            Next y
        Next x
    End With
End Sub

```

Подводя короткий итог можно сказать, что выделенный диапазон можно получить используя объект **Selection**, перевести его в объект **Range**, от которого можно получить данные о местоположении выделенного диапазона, количества выделенных ячеек, а также иметь доступ к отдельным ячейкам используя объект **Range**.

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 32 - Перемещение по ячейкам и информация

Вас может заинтересовать, а как можно сдвинуться влево или вправо назад или вперед от текущей ячейки. Для этого у объекта **Range** есть метод **Offset**, который и позволяет производить подобные действия.

```
Sub Test()  
    Dim cur_range As Range  
    Set cur_range = Range("A1")  
    Set cur_range = cur_range.Offset(1  
    Debug.Print cur_range.Address  
End Sub
```

А вот результат работы. Мы от текущего объекта сдвинулись влево на 1 колонку.

\$A\$2

Если вы хотите узнать максимальные размеры листа, то у Вас есть возможность это сделать используя **UsedRange**. У вас будет объект типа **Range**, из которого вы сможете узнать максимальную колонку или строку.

```
Sub Test()  
    With ActiveSheet  
        Dim cur_range As Range  
        Set cur_range = .UsedRange  
        Debug.Print cur_range.Address  
    End With  
End Sub
```

Адресовать ячейки можно и двумя цифрами по колонки и строке. Это избавляет Вас от утомительного разбора адресов типа **\$A10**. Так как адрес

строка придется её резать и собирать. Использование **Cells(x,y)** очень гибко в использовании и позволяет строить легкие циклы. Пример ниже находит на листе левый верхний угол из всех ячеек с введенными данными и в эту ячейку записывает слово.

```
Sub Test()  
    ' объект Range  
    Dim cur_range As Range  
    ' Весь лист  
    With ActiveSheet  
        Set cur_range = .UsedRange  
        Debug.Print cur_range.Address  
        ' у меня печатает $C$5:$J$5  
        Dim y_min As Integer  
        ' минимальная колонка  
        y_min = cur_range.Columns.ColumnIndex  
        Dim x_min As Integer  
        ' минимальная строка  
        x_min = cur_range.Rows.RowIndex  
        Set cur_range = Range(Cells(x_min, y_min), cur_range)  
        cur_range = "lef up"  
    End With  
End Sub
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

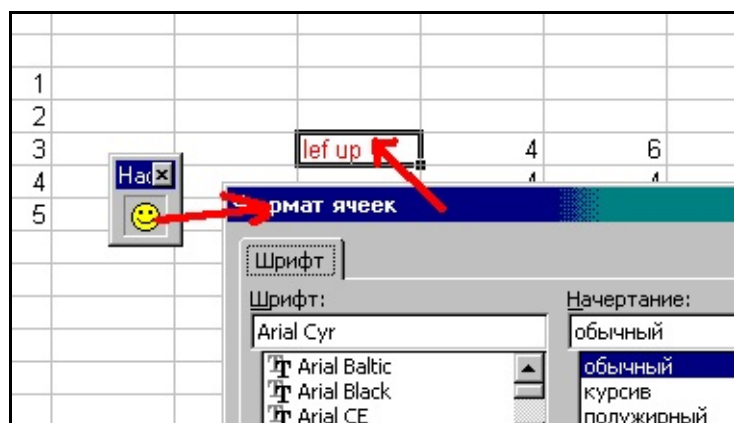
Автор [Каев Артем](#).

## Шаг 33 - Встроенные диалоговые окна

**Excel** имеет несколько встроенных диалоговых окон. Несколько это слабо сказано, их более 200. Предназначены они для облегчения работы и программирования. Например, вашему приложению необходимо вызывать окно диалога для выбора цвета ячейки. Вот код:

```
Sub Test()  
    Application.Dialogs(xlDialogActiveCellFormat) Show  
End Sub
```

А вот результат работы при запуске макроса. Это окно появится и будет изменять свойства выделенного диапазона.



Кроме типа окна далее можно передавать параметры. Вот, например, для открытия **DBF** файла.

```
Sub Test()  
    Application.Dialogs(xlDialogOpen) Show  
End Sub
```

Появится диалоговое окно с предложением выбрать **DBF** файл. Как вы заметили для отображения окна используем метод **Show** и уникальную константу диалогового окна.

С аргументами особый разговор. Во-первых их может быть много. Если нужно оставить аргумент по умолчанию, то используйте вот такую конструкцию "**\*.dbf**", **,TRUE**. Две запятые позволяют пропустить аргумент ( оставить по умолчанию). Так же следует знать, что аргументы нужно задавать строго в определенной последовательности при работе со встроенными диалоговыми окнами.

Аргументы могут быть именованными. Ниже пример аналогичен примеру с двумя запятыми.

```
Sub Test()  
    Application.Dialogs(xlDialogOpen).!  
End Sub
```

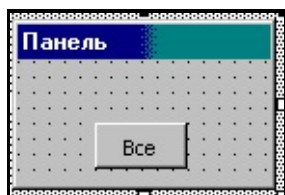
---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 34 - Архитектура программ VBA

Одна из концептуальных идей **Windows** и программирования для **Windows** заключается в том, что объекты обмениваются сообщениями. Именно обмен, получение и обработка сообщения являются смыслом жизни любого объекта. Давайте посмотрим. У нас есть диалоговая панель и кнопка. Например такая.



Эту панель можно создать, показать методом **Show** и убрать методом **Unload**. Между вызовами этих процедур объект живет. То есть получает и обрабатывает сообщения. Например, при нажатии на кнопку.

Посылку сообщения можно рассматривать, как вызов метода соответствующего объекта. Например, Вы нажимаете на кнопку мышкой. Нажатие состоит из кучи сообщений - мышка двигается, клавиша вниз, клавиша вверх и другие. При этом обрабатывается последовательность сообщений и система делает вывод о сообщении более высокого уровня - нажата кнопка. В результате вызовется метод. То есть сообщения нажатия на кнопку вызывает метод **Click** этой кнопки.

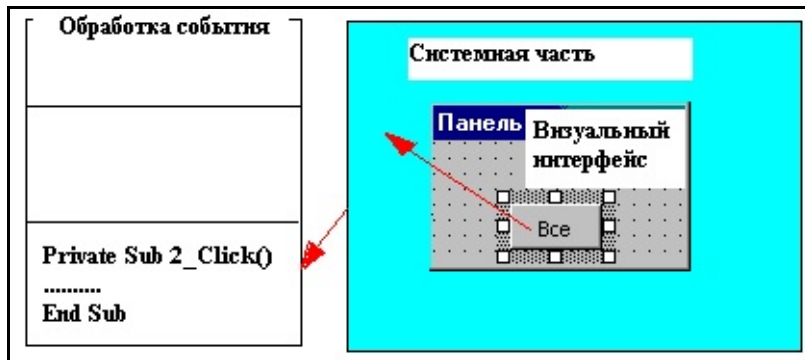
```
Private Sub CommandButton2_Click()  
    .....  
End Sub
```

Вообще-то сообщения примерно так и работают и в реальном мире. Сообщите жене, что у Вас появилась другая женщина и у жены будет вызван соответствующий метод. Реализация этого метода зависит от конструкции объекта жена :-). Или позвоните 03 и сообщите адрес со словами пожар. То же будет реакция. Вообще как в жизни. Только для получения реакции нужно послать сообщение.

Модель **VBA** подразумевает три составляющих:

- Визуальная
- Системная
- Обработчик событий

Посмотрите рисунок ниже.



Визуальная часть это то, что видно на экране, т.е. интерфейс пользователя. Это окна диалога, кнопки, списки и т.д. При работе с программой пользователь постоянно её теребит - нажимает кнопки, двигает окна и еще производит кучу действий. Он использует интерфейсные объекты (элемены управления) для генерации событий. В ответ на это системная составляющая, которая включает в себя:

- средства операционной системы
- средства языка прогарммирования

определяет соответствующее событие и формирует сообщение объекту (вызывает метод объекта). Обработчик событий это код, который будет вызван при возникновении события.

**VBA** для **OFFICE** полностью соответствует этой концепции. Офис предоставляет Вам средства интерфейса, **VBA** реакцию на события. Вы проектируете интерфейс и реакцию. Системная часть Вас не волнует. Это на совести разработчика **VBA** и **OFFICE**.



Рассуждать о преимуществах и недостатках данной системы можно долго. Только идея здесь следующая. Операционная система и реализация среды программирования может меняться (она и меняется 3.1, 95, 98 etc.) , меняется **VBA** ( 95 , 97 etc.), а вроде как ваши программы от этого вообще не зависят. Например, если в следующей версии **WINDOWS** кнопка будет допустим галлографическая, то ваша программа будет с ней работать :-). Вам придется при необходимости добавить новые методы.

Визульный он потому, что код рождается от визуального интерфейса. То есть строите интерфейс потом только код реализации. Бейсик потому, что он и есть Бейсик с дополнительными возможностями. Представляете как далеко смотрели в будущее наши учителя, по школьной программе изучается Бейсик.

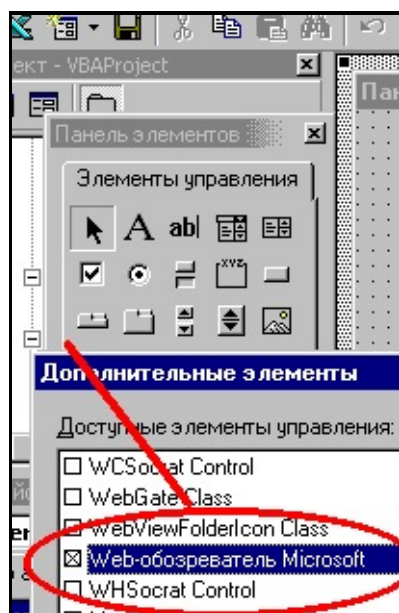
---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

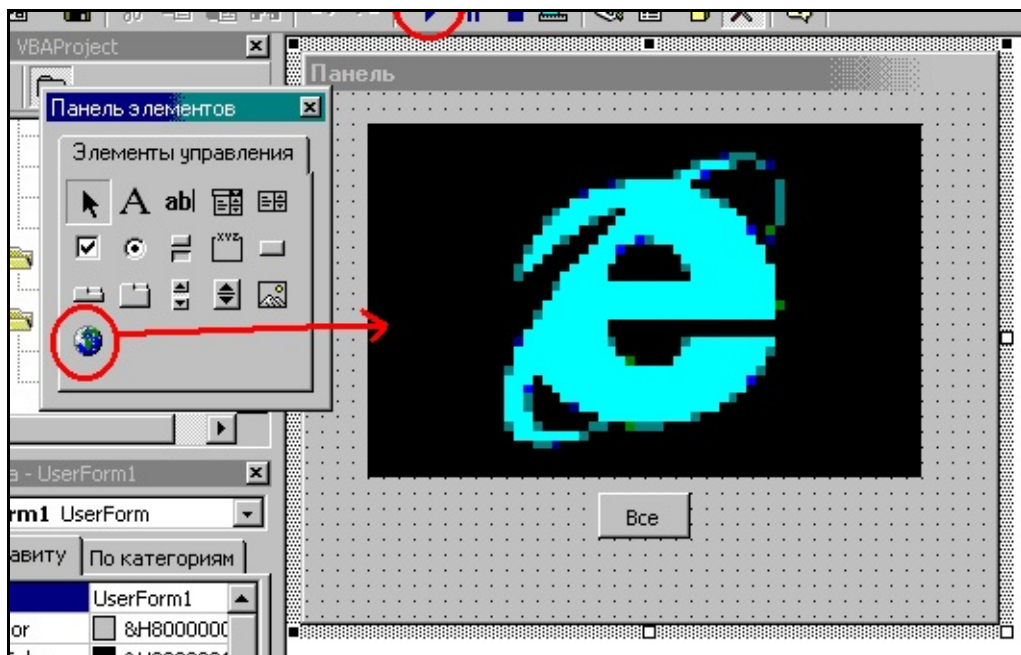
Автор [Каев Артем](#).

## Шаг 35 - Дополнительные компоненты

Когда вы редактируете диалоговое окно, вы видите далеко не все компоненты, которые есть в системе. Для получения полного списка вам необходимо щелкнуть правой кнопкой на Панели инструментов и выбрать "Дополнительные компоненты". Выбрать можно любой, но мне понравился **Web-обозреватель**, его я и подключил.



В панели компонент появится земной шар. Выберите его и поместите на диалоговую панель.



Два раза щелкните на кнопки и привезите код к событию нажатия.

```
Private Sub CommandButton2_Click()  
    WebBrowser1.GoHome  
End Sub
```

Вернитесь к редактированию диалоговой панели. Её тут же можно запустить на исполнение. Смотрите на рисунке обведено кружком. Установите соединение с интернетом. Нажмите на кнопку. Загрузится страница **HOME**. Там про то, что "Добро пожаловать".

Как видите подключение и использование дополнительных компонент дело в принципе несложное. Это только в принципе :-).

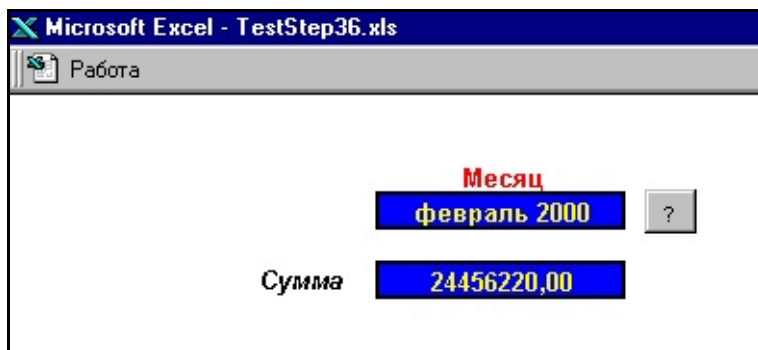
---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 36 - Где хранятся настройки панелей инструментов

Это очень интересный вопрос. Вот пример. Я создал настройки панелей инструментов как на картинке.



Всё это я сохранил в файле. Теперь открывая **Excel** такой вид будет у всех книг. Значит информация о настройках панелей инструментов где-то хранится. Конечно !!!

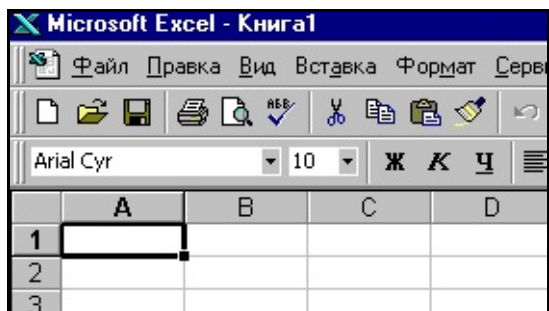
Параметры модифицированных панелей инструментов хранятся в файле.

`WINDOWS\ИМЯ_ПОЛЬЗОВАТЕЛЯ.XLB`

или

`WINNT\ИМЯ_ПОЛЬЗОВАТЕЛЯ.XLB`

Что понимается под именем пользователя? Вот я вхожу в **NT** и к имени **Administrator** ввожу пароль. Поэтому этот файл имеет имя **Administrator.xlb**. Там все настройки. Если вы хотите запускаться с настройками по умолчанию переместите его в другое место и **Excel** сам настроит панели инструментов по умолчанию. Вот смотрите. Сейчас он загружается как на рисунке сверху в смысле панелей инструментов. Переносу файл из каталога **WINNT** на рабочий стол. И вот.



Настраивать панель инструментов можно также из **Visual Basic** используя объект **CommandBar**.

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 37 - Создание приложений с использованием Excel

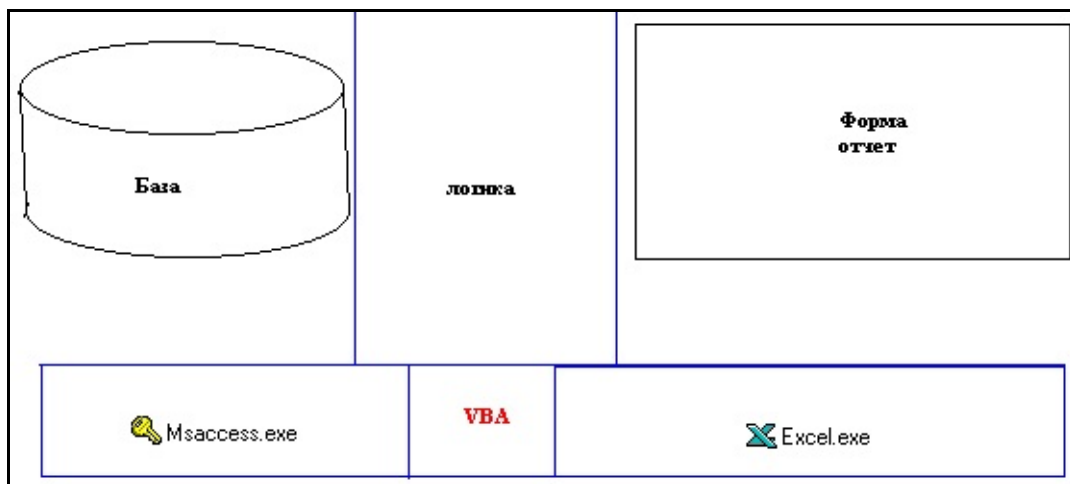
Внимание! Это личный взгляд автора на построение приложений на базе Microsoft Office. Претензии и комментарии принимаются по почте.

Итак, нам нужно построить приложение. Любое приложение состоит из 3 частей.

1. Место хранения информации
2. Интерфейс
3. Логика.

Всё это конечно можно реализовать с помощью **Excel**, вопрос какой ценой? Вообще в этой жизни можно всё. Например, взломать сеть используя только **NotePad**. Можно, пишете в командах процессора, потом переименуйте **txt** в **exe** и готово. Теоритически можно. Вы можете? Я нет. Вот и о чем речь. Сколько и какой ценой.

Итак мой опыт такой. Неправильное использование инструментов ведет к головной боли программистов. Любое приложение можно сделать, например, и в **Excel**, и в **Access**. Но только реально начнете работать то тут стоп. В **Excel** легко создавать формы и отчеты, считать, но хранить данные тяжело. А **Access** нет проблем с хранением, контролем за информацией, но все остальное труднее. Все просто. Каждый инструмент для своей задачи. Вот мой взгляд.



Как видите данные хранятся в **Access** формы и отчеты в **Excel**, логика реализуется на **VBA**. Обратите внимание, что эта модель не чистая. Все таки часть ответственности за логику ложится и на **Access** и на **Excel**. Например, в **Access** можно установить фильтры на ввод, построить запросы. В **Excel** проводить расчеты. **VBA** является связующим звеном между этими программными продуктами.

Связь между **Access** и **Excel** можно организовать по разному. Например, на основе **DAO**. Но встает вопрос, чей **VBA**, то есть какого программного продукта ? Я склоняюсь к **Excel**. Вот почему. Работа происходит так:

работа с формой  
 нужна информация  
 запрос с базе  
 получение результатов  
 возврат на форму

То есть основное количество логических операций производится в интерфейсе. Поэтому удобно пойти от **Excel** туда и поместить код **VBA**. Все, что нужно сделать это научить тесно взаимодействовать **Excel** и **Access**.

---

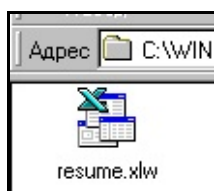
[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 38 - Зачем нужна рабочая область

Без тебя лето зима  
Без тебя метель в июле  
А Студио.

Рабочая область - это набор файлов **Excel**. Она позволяет открывать эти файлы одновременно. Впрочем-то это специальный файл, который содержит информацию о том какие рабочие книги должны быть открыты. Его расширение **XLW**. А выглядит он вот так:



Рабочая область имеет смысл, если у Вас есть несколько рабочих книг связанных между собой. Давайте попробуем. Создавайте рабочий каталог с именем **TestWorkspace**. Теперь создадим файл **Excel** с именем **Test1** и поместим его в рабочий каталог. Давайте занесем информацию. Смотрите ниже. Три участка с фамилиями. Красным это сумма полученная автосуммированием.

	А	В	С	Д	Е	F	G	Н
1								
2								
3								
4		Участок 1			Участок 2			Участок 3
5	Петя	123		Оля	44		Бобик	4
6	Коля	233		Маша	33		Шарик	5
7	Вася	445		Наташа	55		Рекс	6
8	Дима	555		Даша	66		Бим	66
9		1356			198			81
10								
11								
12								

Сохраните этот файл и, не закрывая его, создайте новый. Дайте ему имя **Test2.xls** и сохраните в тот же каталог. Теперь мы занесем в него



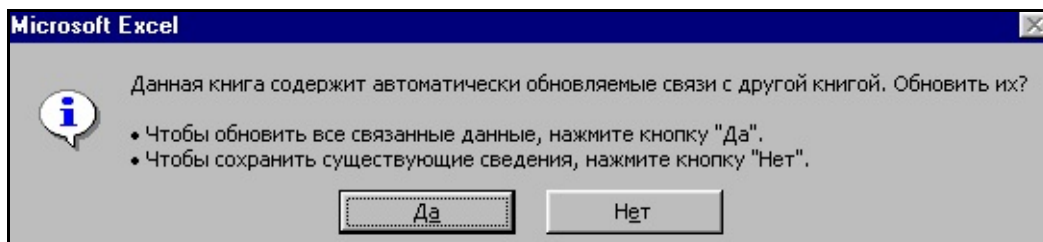
информацию ссылаясь на суммы в первой книге (**Test1.xls**).

	A	B	C
1			
2			
3			
4	Работники (Участок 1)	1356	
5	Работницы (Участок 2)	198	
6	Халявщики (Участок 3)	81	
7		1635	

Теперь сохраним рабочую область и проведем эксперименты. Выбираем меню "Файл -> Сохранить рабочую область". И сохраняем в тот же каталог с именем **Test**. Закрываете **Excel** в вашем каталоге должно быть три файла.

[Test1.xls](#)  
[Test2.xls](#)  
[test.xlw](#)

Экспериментируем. Откройте **Test2.xls**. Вы получите сообщение:



Это нормально. Связи то есть. А вот теперь закройте все и откройте рабочую область. Всё пройдет без сообщений. Конечно откроются сразу все файлы в рабочей области. Это удобно если предполагается обмен данными между многими книгами. При этом всё должно быть динамично.

---

[Загрузить проект](#) | [Предыдущий Шаг](#) | [Следующий Шаг](#)

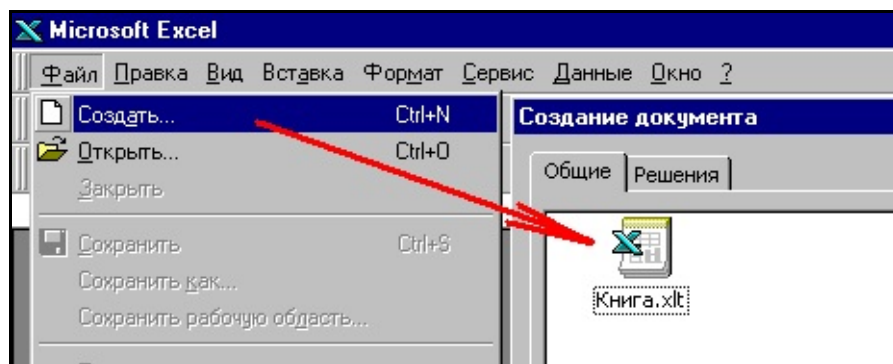
Автор [Каев Артем](#).

## Шаг 39 - Автозапуск и шаблоны

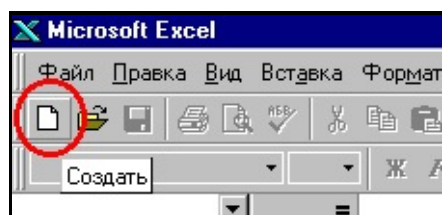
Понимание когда и откуда создается пустой лист и при каких условиях - это залог к автоматизации приложений. Например, вы установили параметры страницы, макросы, шрифты и цвета и хотите, чтобы автоматически создаваемая книга их имела. Это понятие шаблона. Только создается книга двумя разными путями.

- Из каталога шаблонов
- Из каталога автозапуска

Из шаблонов:



Папки автозапуска:



Весь прикол в том, что эти папки разные :-))) Все равно Вы не захотите, чтобы Ваши документы создавались всегда одинаково !!!

C:\Program Files\Microsoft Office\Шаблоны  
C:\Program Files\Microsoft Office\Office\X

Давайте убедимся, что это разные вещи. Создайте шаблон скажем с желтой верхней строкой и сохраните в шаблоны, тоже самое сделайте только с красной и сохраните в папку автозапуска. Создавайте шаблоны с именами **Книга.xlt**. Закройте **Excel**, а теперь попробуйте два варианта и результат будет разный. Это важно. Важно особенно при программировании, чтобы быть уверенными, что все элементы в книге есть.

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 40 - О многозадачности Windows и циклах

Как Вы знаете **Windows 9x** является многозадачной средой. Это везде пишется. Но на самом деле это далеко не совсем так. То есть в ней нет четкой установки приоритетов. Не верите ? Создайте макрос в **Excel** и запустите вот этот пример.

```
Sub Test()  
    For x = 1 To 10000000000000  
        Debug.Print x  
    Next x  
End Sub
```

Любая работа в этот момент будет проблематична. Это связано с проблемами еще от **Windows 3.1**, тогда при программировании от Вас требовали периодически особенно в процессе длительных циклов передавать управление операционной системе. **VBA** до сих пор не избавлен от этой проблемы. Вам все равно надо это делать. Делается это с помощью **DoEvents**.

Используйте функцию **DoEvents** для передачи управления операционной системе каждый раз прохода цикла. Но она может быть полезна и при дисковых операциях ввода вывода, операциях с **DDE**. Давайте изменим наш пример.

```
Sub Test()  
    For x = 1 To 10000000000000  
        DoEvents  
        Debug.Print x  
    Next x  
End Sub
```

Если Ваша программа тормозит выполнение других программ вспомните

об этом шаге.

---

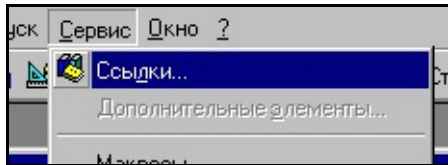
[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

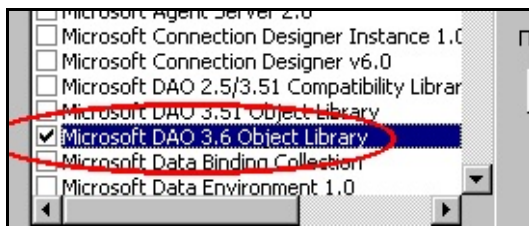
## Шаг 41 - Подключаем DAO

Следующая серия шагов будет посвящена проб. Задача сделать форму Excel, которая будет из цены в долларах оперируя информацией из

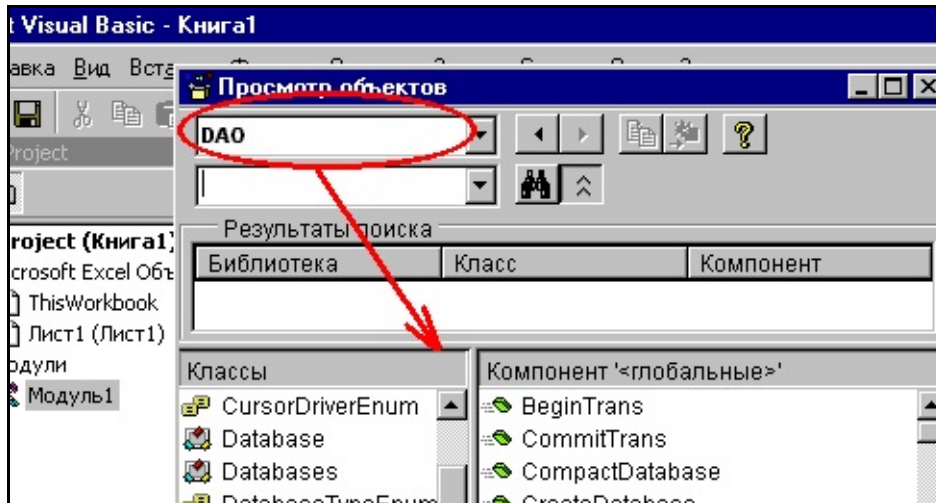
В Excel много методов работы с базами данных. Давайте попробуем DAO для того, чтобы получить доступ к классам DAO нам необходимо их подключить. Это делается из меню "Ссылки":



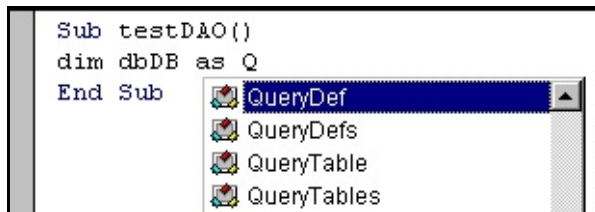
Откроется диалоговое окно, в котором нам надо найти **DAO Object Library**.



С этого момента вы можете многое. Например, просмотреть список классов и их свойств и методов. Сразу скажу, что это надежнее документации. Там есть то, чего нет в описаниях и помощи. Зайдите в меню "Вид -> Просмотр объектов" и выберите **DAO**.



Теперь мы можем использовать классы **DAO**.



---

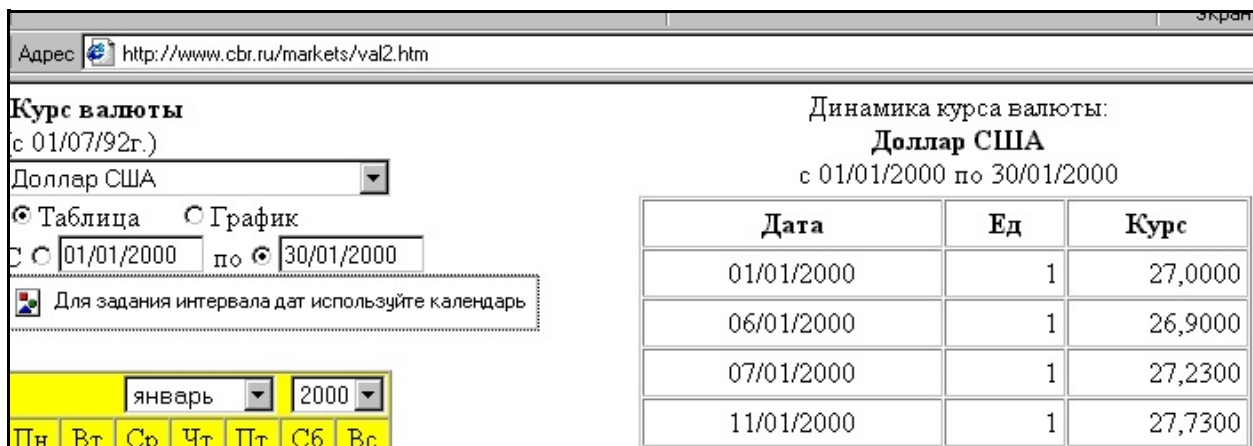
[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 42 - Готовим данные

Для того, чтобы быть ближе к жизни я пошел на сайт Центрального Банка России за курсом доллара. Вот его адрес, чтобы долго не ходить (<http://www.cbr.ru/scripts/daily.asp>)

А вот как выглядит страница с курсами:



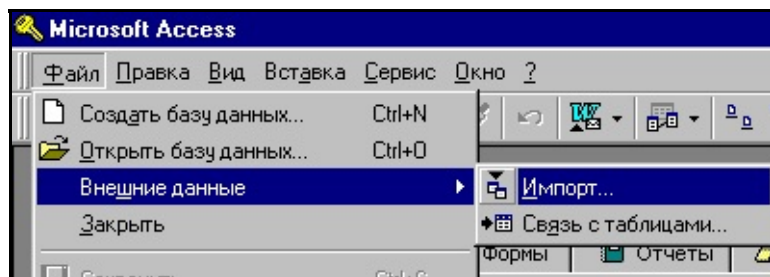
Дата	Ед	Курс
01/01/2000	1	27,0000
06/01/2000	1	26,9000
07/01/2000	1	27,2300
11/01/2000	1	27,7300

Я просто выделил и скопировал в текстовый файл через буфер обмена. Он примерно такой с именем **Curs.txt**. Он есть в проекте если кому лень.

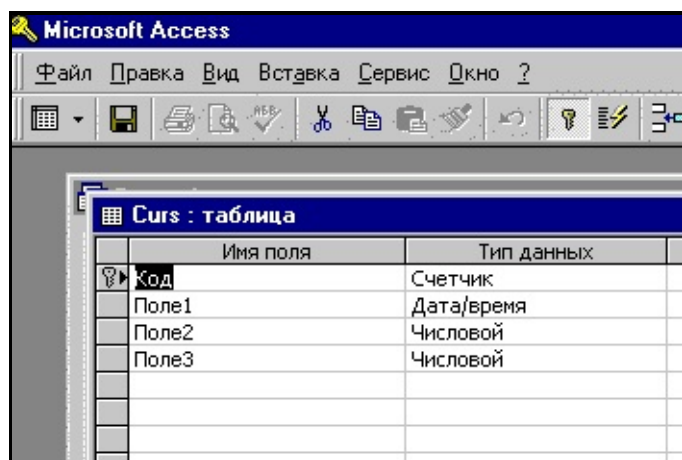
```
01/01/2000 1 27,0000
06/01/2000 1 26,9000
07/01/2000 1 27,2300
11/01/2000 1 27,7300
12/01/2000 1 28,4400
.....
```

Теперь мы из этого текстового файла сделаем базу данных. Запустим **Access** создадим новую БД с именем **curs** и меню "Импорт":





А дальше "Текстовый файл" и "фиксированной ширины". В результате появится таблица со следующей структурой и заполненными значениями.



Для особенно ленивых этот файл есть в проекте **curs.mdb**. Только если вы настолько ленивы зачем вообще читаете ???

---

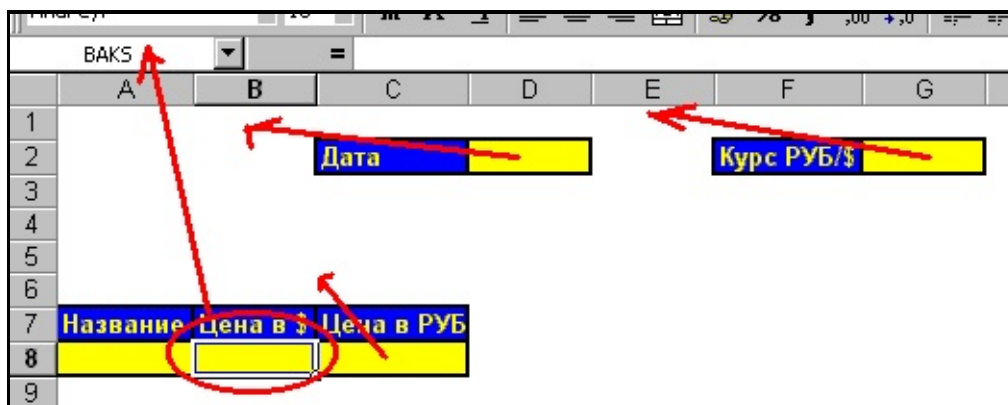
[Загрузить проект](#) | [Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

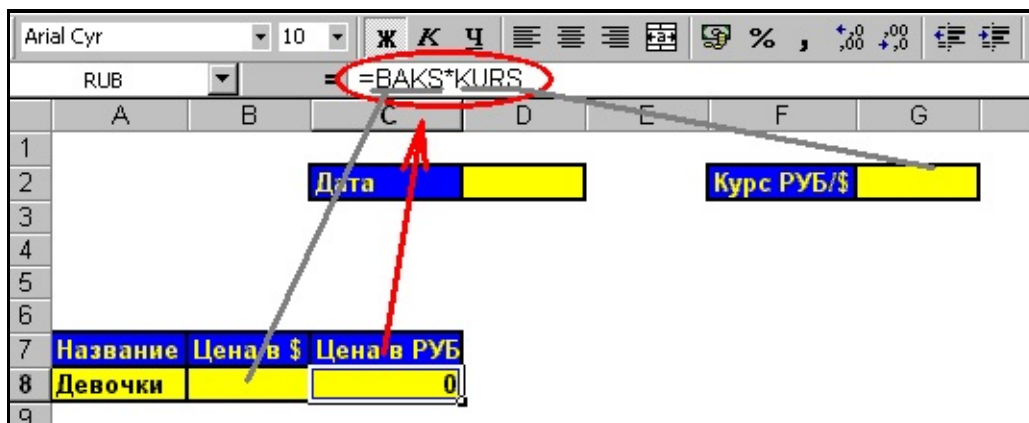
## Шаг 43 - Готовим форму

Итак, нам необходимо подготовить форму, к которой мы будем обращаться. Конечно это лист **Excel**. Запустите **Excel**, создайте файл с именем **TestCurs** и оставьте на нем один лист. Теперь на этом листе напишем "Цена товара в \$" и "Цена товара в РУБ", также отдельно "Курс РУБ за \$". Нужна и ячейка "Дата". И дадим имена ячейкам.

Цена в рублях      RUB  
Цена в \$                      BAKS  
Курс                      KURS  
Дата                      DATES



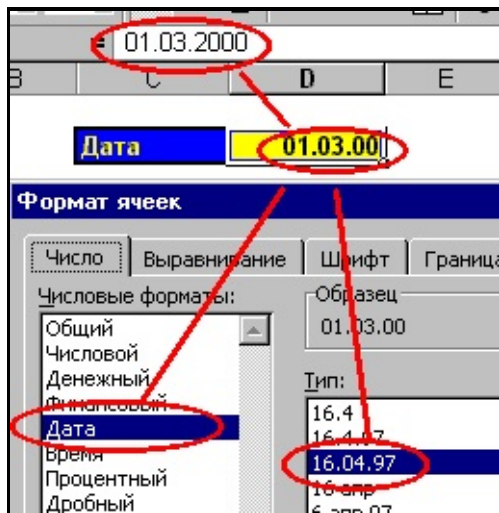
Теперь дело за формулами. Идея ясна как белый день. Стоимость руб = Стоимость в \$ \* курс. Вот это и запрограммируем:



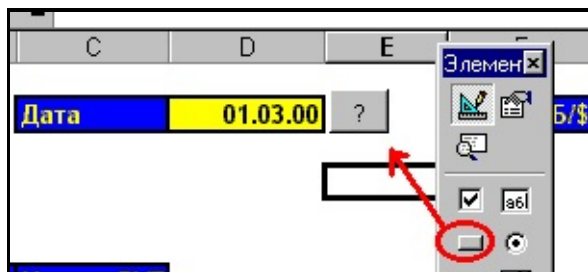
Вводить мы будем цену в долларах, выбирать дату и видеть цену в рублях. Цену в долларах ввести не тяжело :-))). Все остальное посчитается, если будет курс. Курс привязан к дате. То есть мы должны.

- Ввести дату
- Получить курс
- А дальше вводить цену в долларах

Вот теперь надо все подготовить для ввода даты. Надо задать, что это ячейка даты. Выделите её пойдите в "Формат", дальше "Ячейки", потом установите тип ячейки "Дата".



После того как введена дата нужно будет получить курс. Для этого поместим кнопку на лист. Нам нужно в меню "Вид -> Панели инструментов -> Элементы управления" выбрать кнопку и поместить её рядом с датой. Нажимая на неё мы будем получать курс.



Двойной щелчок создаст макрос. Вот и все на этот шаг. Все готово к программированию. Эта книга есть в проекте, если у Вас что-то не получилось.

---

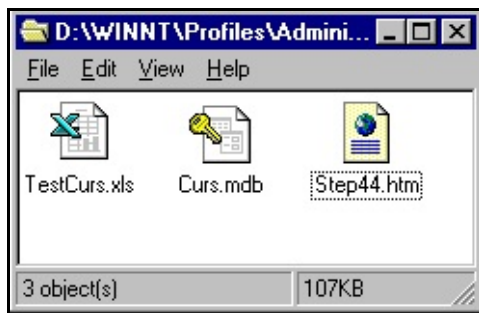
[Загрузить проект](#) | [Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 44 - Считаем

Голос тихий таинственный  
Где ты милый единственный  
Алсу.

Итак, нам нужно связывать **Excel** и **Access**. Где находится **Excel** мы знаем, а вот где файл **MDB** нужно знать и указывать. Но этих проблем можно избежать, если применить такой ход. Если **mdb** будет в том же каталоге, что и **XLS**, то можно создать функцию, которая будет получать этот путь.



Итак мы её поместим в отдельный модуль для этого его еще нужно создать.  
А вот код.

```
Function stDBGetPath() As String
    Dim stTemp As String
    ' взять путь нахождения активной к
    stTemp = ActiveWorkbook.Path
    ' прибавить имя базы данных
    stDBGetPath = stTemp + "\" + "curs
End Function
```

Вот теперь можно открывать базу данных используя объекты **DAO**. Вот код:

```
Private Sub CommandButton1_Click()
    '----- объявление переменных
```

```

' переменная типа базы данных
Dim dbAccess As Database
' переменная типа набор записей
Dim reRecordSet As Recordset
' здесь будет SQL запрос
Dim stSQL As String
' переменная типа даты
Dim daDate As Date

' а вдруг ошибка
On Error GoTo ErrorsDB

' ----- получаем данные из ячейки
daDate = Range("DATES").Value
' ----- работа с базой -----
' откроем базу данных
Set dbAccess = OpenDatabase(stDBGe
' Строим SQL запрос
stSQL = "SELECT * FROM[Curs] WHERE
' получаем набор значений
Set reRecordSet = dbAccess.OpenRe
' если данные получены тогда занесем
If (reRecordSet.RecordCount > 0) Then
    ' поместить значение в ячейку
    Range("KURS").Value = reRe
Else
    MsgBox "Not Found"
End If

' закрываем набор записей
reRecordSet.Close
' закрываем базу данных
dbAccess.Close

```

```
' все в норме конец  
GoTo Ends  
' ошибка где-то однако
```

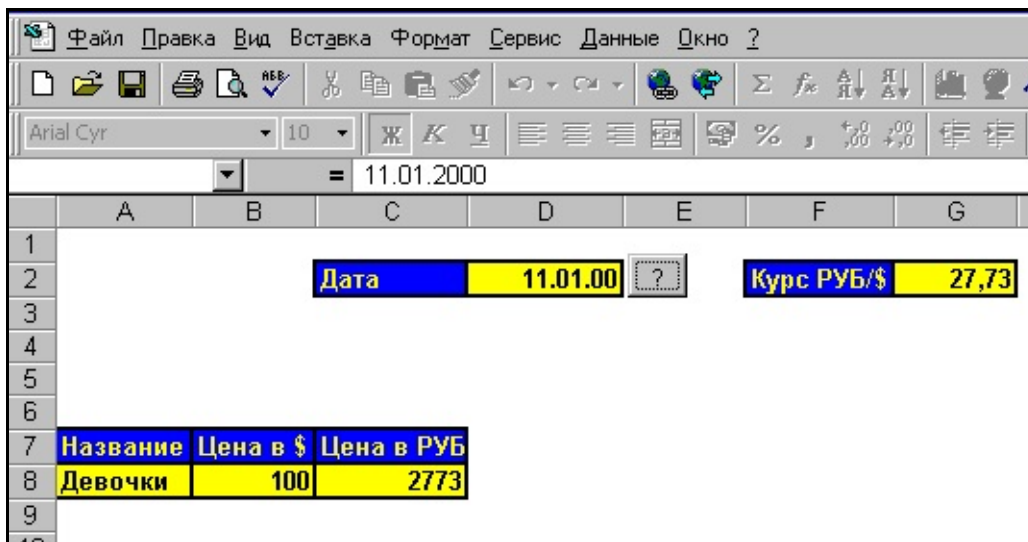
```
ErrorsDB:
```

```
MsgBox "Произошла ошибка"
```

```
Ends:
```

```
End Sub
```

Поле с датой в файле **Curs.mdb** нужно перевести в текстовый формат. Теперь испытания. Введите дату и цену в \$ и нажмите кнопку рядом с датой. Если дата есть, то курс поменяется, иначе получите сообщение, что нет данных **Not Found**. Поменяйте дату и опять нажмите кнопку. Все пересчитается. Вот пример работы:



---

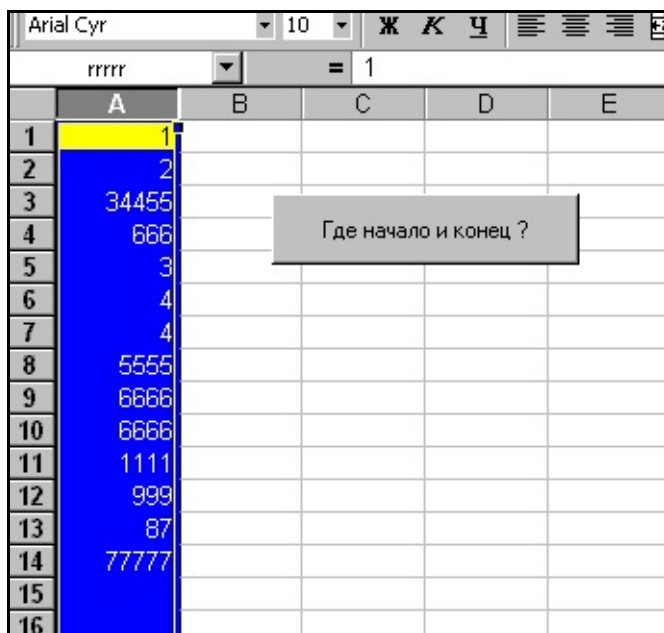
[Загрузить проект](#) | [Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 45 - Начало и конец данных

Если мы захотим изменить наш пример в плане, что нам цена нужна не только одного товара, но и нескольких скажем картошка, лук и апельсины, то возникнет вопрос как это сделать. Ответ это именованный диапазон. Тогда как определить его начало и конец ? В том смысле где находятся цифры ? Вот этим мы и займемся в этом шаге.

Создаем новую книгу и колонку **A** переименовываем в диапазон **rrrrr**.



Так как диапазон есть, теперь к нему можно применять всякие там методы. Выделяем его **Range**, у **Range** есть метод **End** - мол где конец или начало :-))) , а у **End** есть адрес вообще-то говоря смотрите !!!

```
Sub TestRange()  
    Dim r As Range  
    Set r = Range("rrrrr")  
    MsgBox (r.Columns.End(xlUp).Address)  
    MsgBox (r.Columns.End(xlDown).Address)  
End Sub
```



Вот теперь мы и знаем адрес первой и последней ячейки !!! . Привяжем вызов данной процедуры к нажатию кнопки.

```
Private Sub CommandButton1_Click()  
    TestRange  
End Sub
```

---

[Загрузить проект](#) | [Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 46 - Доступ к одинаковым элементам управления

Артем привет!

Есть вот такой код:

```
str1.Cells(kod_filiala + 6, 4) = V  
str1.Cells(kod_filiala + 6, 5) = V  
str1.Cells(kod_filiala + 6, 6) = V  
str1.Cells(kod_filiala + 6, 7) = V  
str1.Cells(kod_filiala + 6, 8) = V  
str1.Cells(kod_filiala + 6, 9) = V  
str1.Cells(kod_filiala + 6, 10) = V  
str1.Cells(kod_filiala + 6, 11) = V
```

Возможно ли все эту писанину заменить на то

```
For i=1 to n  
    Условия .....  
end if
```

т.е. чтобы ТекстБоксы перебирались как массивы.  
С Уважением, Владимир.

\*\*\*\*\*

With best wishes, Vladimir(Владимир)  
My E-mail [stilvlad@chat.ru](mailto:stilvlad@chat.ru) (text only, koi  
[stilvlad@mail.ru](mailto:stilvlad@mail.ru) (all formats).

My Web: <http://stilvlad.chat.ru>

\*\*\*\*\*

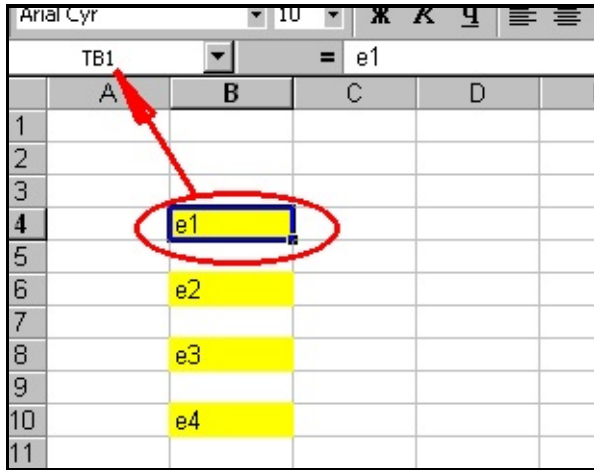
\*Пожалуйста цитируйте всю переписку со мною

Чтож давайте попробуем. Создаем книгу. На ней расположим именованные ячейки.

ТВ1

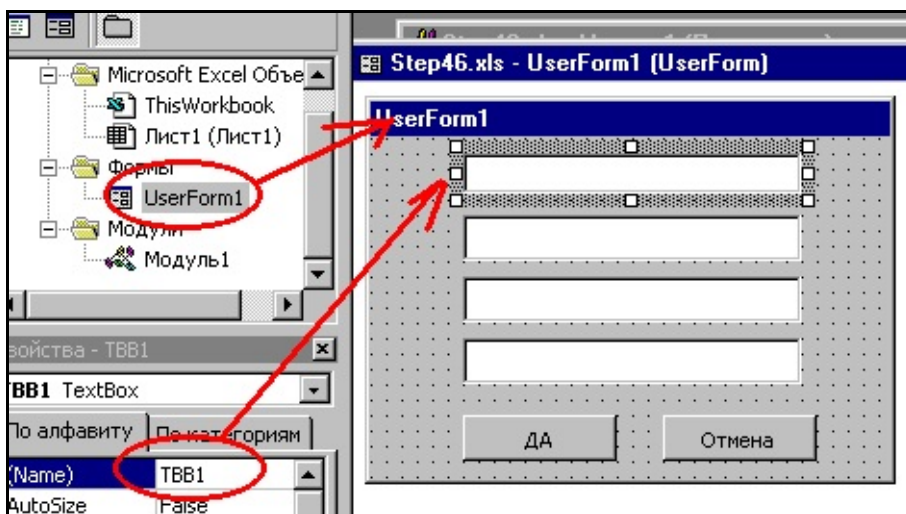
ТВ2

TB3  
TB4



Создали форму и на ней элементы **TextBox**. Внимание, сначала создавайте **TextBox**, а потом кнопки, это позволит избежать проверки типов для **Controls**. Имена:

TBB1  
TBB2  
TBB3  
TBB4



Код по кнопке "ДА":

```

Private Sub CommandButton1_Click()
    ' объект элемент управления
    Dim objObject As Control
    ' объект диапазон
    Dim raRange As Range
    ' строка с адресом диапазона
    Dim stAdders As String
    ' цикл по всем элементам управления
    For x = 0 To UserForm1.Controls.Count - 1
        ' присвоить объекту элемент управления
        Set objObject = UserForm1.Controls(x)
        ' создать адрес ячейки
        stAdders = "ТВ" + LTrim(Str(x))
        ' получить диапазон
        Set raRange = Range(stAdders)
        ' присвоить ему значение и адрес
        raRange.Value = objObject.Value
    Next x
    ' закрыть форму
    Unload Me
End Sub

```

Код кнопки "нет":

```

Private Sub CommandButton2_Click()
    Unload Me
End Sub

```

Запускной макрос:

```

Sub TestForm()
    UserForm1.Show
End Sub

```

Вот и все. Попробуйте. Можете загрузить проект и посмотреть.

---

[Загрузить проект](#) | [Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 47 - Свойства документов MSOffice

Аналогично свойству **Tag** форм и элементов управления, свойства документа позволяют программисту сохранять необходимую информацию (о возможности использования свойств, точнее их значений, для поиска документов здесь упоминать не будем - это пользовательская возможность). При необходимости эту информацию можно легко извлечь и использовать.

В VBA для MSOffice определены два типа свойств документов: встроенные (BuiltIn) и пользовательские (Custom). Оба типа свойств организованы в коллекции, соответственно `BuiltInDocumentProperties` и `CustomDocumentProperties`, относящиеся к объекту(ам) `Document` и `Template`. Нумерация элементов коллекций начинается с единицы.

Встроенные свойства документов автоматически поддерживаются соответствующими приложениями, и без необходимости модифицировать их не стоит, можно считывать и использовать.

Гораздо больший интерес для программиста представляют Custom свойства документа - именно их можно использовать для хранения нужной информации, связанной с документом. Достаточно удобно то, что можно использовать разные типы свойств: строки, числа, даты, и логические поля.

Добавить нужное свойство (здесь и далее будем полагать, что работаем со свойствами активного документа) можно, используя метод **Add**:

```
ActiveDocument.CustomDocumentProperties.Add  
Name:=PropertyName, _  
LinkToContent:=False, _  
Value:="", _  
Type:=msoPropertyTypeString
```

В этой инструкции:

- **Name:=PropertyName** - имя свойства, может быть строковым выражением или переменной.

- **LinkToContent:=False**, булево значение, определяющее связь с элементами контейнера в самом документе. Используя False, будет создано статическое свойство, именно оно и интересно. Если установить в True, то надо дополнительно определить LinkSource, указывающий на соответствующий объект документа.
- **Value:=""**, собственно значение, необязательный параметр.
- **Type:=msoPropertyType**, тип свойства, определены следующие константы типов (в скобках приведены числовые значения констант, тип Long):
  - *msoPropertyTypeBoolean* (2),
  - *msoPropertyTypeDate* (3),
  - *msoPropertyTypeFloat* (5),
  - *msoPropertyTypeNumber* (1), Чем по сути различаются два последних типа - не совсем ясно, по крайней мере оба они способны хранить double число и в стандартном диалог-боксе MSWord у них один тип - Number;
  - *msoPropertyTypeString* (4).

Обязательными элементами являются имя, тип и LinkToContent, значение можно не задавать.

Для считывания некоторого свойства просто пишем:

```
varProperty = ActiveDocument.CustomDocumen
```

где **Name** имя свойства или номер в коллекции.

Все было бы так просто, если бы не одна маленькая неприятность (по крайней мере, в MSOffice97 это так, а с ним еще долго будут работать пользователи). Неприятность заключается в том, что нет возможности прямо проверить, создано ли уже свойство с некоторым именем, а обращение к пустому свойству или попытка создания свойства с именем, совпадающим с именем уже определенного, вызывает ошибку (коды соответственно 5 и -2147467259 - не удивляйтесь, а распечатайте err.number!). Но такое поведение можно с легкостью использовать, написав свой обработчик ошибок.

Итак, необходимо присвоить пользовательскому свойству **Test** значение

переменной **strString**, при этом не известно, существует ли в данный момент пользовательское свойство (даже если вы его уже когда-то определили, то ведь пользователь мог его удалить, значит такая ситуация является общей, а не частной). Вот фрагмент кода для решения такой задачи.

```
Dim strString as String

strString = "тестовое значение пользовател
On Error GoTo AddCustomProperty
ActiveDocument.CustomDocumentProperties.It
On Error GoTo 0

Exit Sub
' ОБРАБОТЧИК(И) ОШИБОК
AddCustomProperty:
    Select Case Err.Number
    Case 5
        ' ЭТОТ номер ошибки возник
        ' в свойство, которое пока
ActiveDocument.CustomDocum
Name:=PropName, LinkToCont
Resume ' возвращаем управл
    Case Else
        ' Вывод сообщений о других
MsgBox(Err.Number & Chr(13)
Exit Sub

End Sub
```

Аналогично, считывание значения пользовательского свойства производится так (тоже ведь никогда нельзя быть уверенным в том, что свойство существует):

```
Dim varProperty as Variant
```



```

On Error GoTo ReadCustomProperty      ' |
strString = ActiveDocument.CustomDocumentP
On Error GoTo 0                        ' |
.....

Exit Sub
' ОБРАБОТЧИК(И) ОШИБОК
ReadCustomProperty:
    Select Case Err.Number
    Case 5
        ' ЭТОТ номер ошибки возник
        ' свойство, которое пока не
        varProperty = "" ' или "Nu
        Resume Next      ' |
        ' кстати, |
    Case Else
        ' Вывод сообщений о других
        MsgBox(Err.Number & Chr(13)
    Exit Sub

End Sub

```

Существенно, что при считывании значения свойства производится автоматическое приведение типа (если оно возможно), и считывание, например, числа в строковую переменную не приводит к ошибке.

Если задан номер пользовательского свойства, то легко можно узнать все его свойства (извините за каламбур), для этого достаточно написать (пример цикла для всех свойств):

```

For Each prop In ActiveDocument.CustomDocu
    With prop
        MsgBox .Name & "= " & .Val
        "Application" & " :

```

```
"Creator" & " = " & ,  
"Parent" & "= " &
```

End With

Next

Для всей коллекции пользовательских свойств легко получить число определенных свойств:

```
ActiveDocument.CustomDocumentProperties.Count
```

Очевидно, что индекс в цикле перебора всех свойств может пробегать от 1 до этого значения.

Для ситуации, когда надо создать свойство, можно тоже написать обработчик ошибок, но логика его работы уже будет сильно зависеть от функциональности программы, в которой он используется. Поэтому он здесь не приводится, однако информации для его написания достаточно. Повторим, что код ошибки для ситуации повторного определения свойства с некоторым именем, равен -2147467259 (знак минус!, впрочем, "правильный" код 440).

Написал и прислал шаг [Dmitri Drozdov](#).

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 48 - Встроенные свойства документов MSOffice

Хорошо писать прикладные программы для MSWord - вставил куда-либо поле **{AUTOR}** и получил информацию о том, кто автор документа (иногда, правда, это делать опасно - всплывают прелюбопытные подробности). А в других программах MSOffice? Сколько, например символов в таблице Excel? Так просто не получится... :-)

Для этой задачи может помочь коллекция `BuiltInDocumentProperties`. В MSOffice97 определено 30 (а не 28, как в документации) свойств, названия которых приведены в таблице. Все эти свойства могут быть прочитаны в любом документе MSOffice, вне зависимости от того, какая программа документ породила. Кроме чтения программным путем, MSWord (это уже упоминалось выше) позволяет вывести значение свойств документа в сам документ при помощи полей, но это к программированию на прямую не относится.

	Название свойства	Назначение
1	Title	Заголовок
2	Subject	Предмет
3	Author	Автор (создавший документ)
4	Keywords	Ключевые слова
5	Comments	Комментарии
6	Template	Шаблон документа
7	Last Author	Тот, кто последний сохранил документ
8	Revision Number	Число входов для редактирования
9	Application Name	Название приложения, обрабатывающего документ
10	Last Print Date	Дата и время последней печати
11	Creation Date	Дата создания
12	Last Save Time	Дата и время последнего сохранения
13	Total Editing Time	Общая продолжительность редактирования (минуты)

14	Number of Pages	Число страниц
15	Number of Words	Число слов
16	Number of Characters	Число символов
17	Security	Секретность
18	Category	Категория
19	Format	Формат
20	Manager	Менеджер
21	Company	Компания
22	Number of Bytes	Размер файла в байтах в момент последнего сохранения
23	Number of Lines	Число строк
24	Number of Paragraphs	Число абзацев
25	Number of Slides	Число слайдов (определено в PowerPoint, в остальных, вероятно, не используется)
26	Number of Notes	Число заметок к слайдам (определено в PowerPoint, в остальных, вероятно, не используется)
27	Number of Hidden Slides	Число скрытых слайдов (определено в PowerPoint, в остальных, вероятно, не используется)
28	Number of Multimedia Clips	Число клипов (определено в PowerPoint, в остальных, вероятно, не используется)
29	Hyperlink Base	База гиперссылок
30	Number of Characters (with spaces)	Общее число символов, включая пробелы

Стоит иметь в виду, что при обращении к значению свойства, которое для данного документа не определено, также возникает ошибка выполнения с кодом -2147467259 (знак минус!), что требует аккуратной работы (см. примеры обработчиков ошибок в предыдущем шаге). Также приводит к ошибке попытка записи в свойство, которое данным приложением не поддерживается, что можно установить методом проб и ошибок. Но имена выводятся для всех свойств правильно. Практически полезной может

оказаться такая процедура:

```
Sub test()  
  
On Error GoTo NotDefined  
' В следующей строке ActiveDocument стоит  
For Each prop In ActiveDocument.BuiltInDoc  
    With prop  
        PName = .Name  
        PValue = .Value  
        PType = .Type  
        Debug.Print PName & " = " ,  
    End With  
Next  
Exit Sub  
' ОБРАБОТЧИК ОШИБОК  
NotDefined:  
PValue = "Value not defined"  
Resume Next  
End Sub
```

Ее польза в том, что можно исследовать, какие свойства определены для данного типа документов, не появились ли новые свойства с выходом новой версии MSOffice (и не потерялись ли старые :-), какие у них типы. Тип выводится в виде целого числа. Результат ее работы (копия **Debug Window**):

```
Title = Свойства документов MSOffice [4]  
Subject = [4]  
Author = DD [4]  
Keywords = [4]  
Comments = [4]  
Template = Normal [4]  
Last author = DD [4]
```

Revision number = 4 [4]  
Application name = Microsoft Word 8.0 [4]  
Last print date = Value not defined [3]  
Creation date = 17.03.00 14:13:00 [3]  
Last save time = 17.03.00 15:06:00 [3]  
Total editing time = 67 [1]  
Number of pages = 3 [1]  
Number of words = 1183 [1]  
Number of characters = 7702 [1]  
Security = 0 [1]  
Category = [4]  
Format = [4]  
Manager = [4]  
Company = SBC [4]  
Number of bytes = 45056 [1]  
Number of lines = 235 [1]  
Number of paragraphs = 161 [1]  
Number of slides = Value not defined [1]  
Number of notes = Value not defined [1]  
Number of hidden Slides = Value not defined [1]  
Number of multimedia clips = Value not defined [1]  
Hyperlink base = [4]  
Number of characters (with spaces) = 8995

Соответствие числа и предопределенных констант типов - ниже:

1	msoPropertyTypeNumber
2	msoPropertyTypeBoolean
3	msoPropertyTypeDate
4	msoPropertyTypeString
5	msoPropertyTypeFloat

Свойства, недоступные в данном приложении, будут иметь вид: **Number of slides = Value not defined [1]**

Написал и прислал шаг [Dmitri Drozdov](#).

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 49 - Связывание макроса с кнопкой на панели инструментов

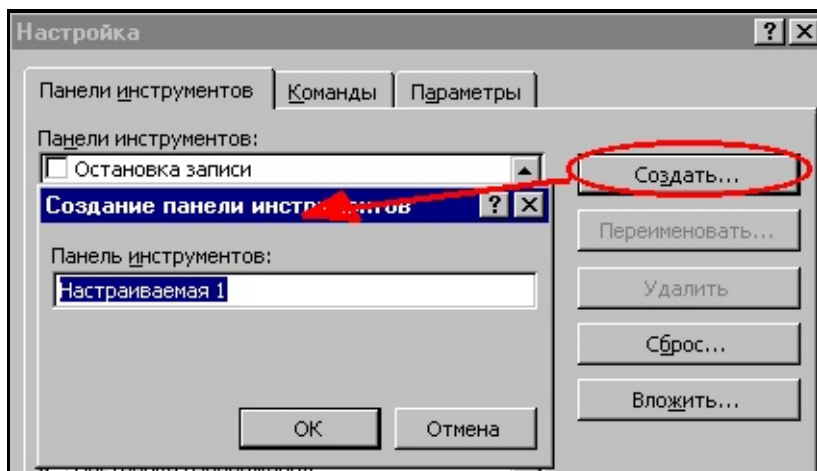
Создавая интерфейс Ваш код на **VBA** в виде макросов можно разместить для доступа пользователей в несколько позиций:

- Меню
- Панели инструментов
- Кнопки формы

Панели инструментов являются наиболее удобными и быстрыми для доступа. Для начала давайте создадим простой макрос. Ну хотя бы вот такой:

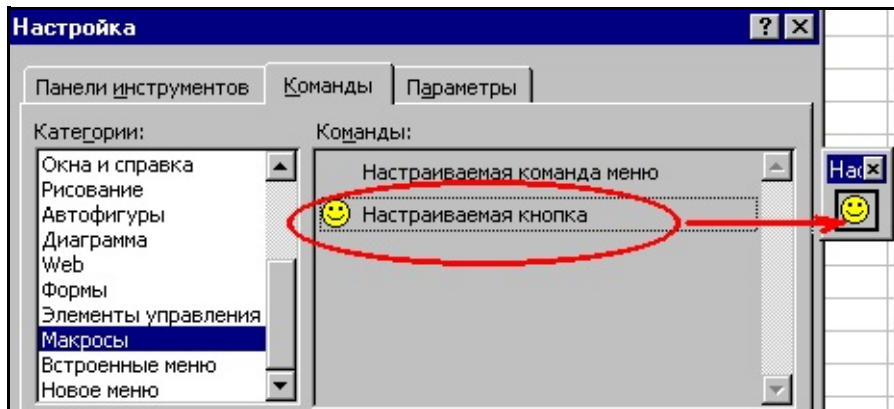
```
Sub Test()  
    MsgBox "Hello"  
End Sub
```

Теперь у нас есть, что связывать. Создать свою панель инструментов можно через меню **Сервис - Настройка**. У Вас по вызову этого пункта меню появляется диалоговое окно с тремя вкладками. Нас интересует сейчас "Панели инструментов", на которой находится кнопка **Создать**.

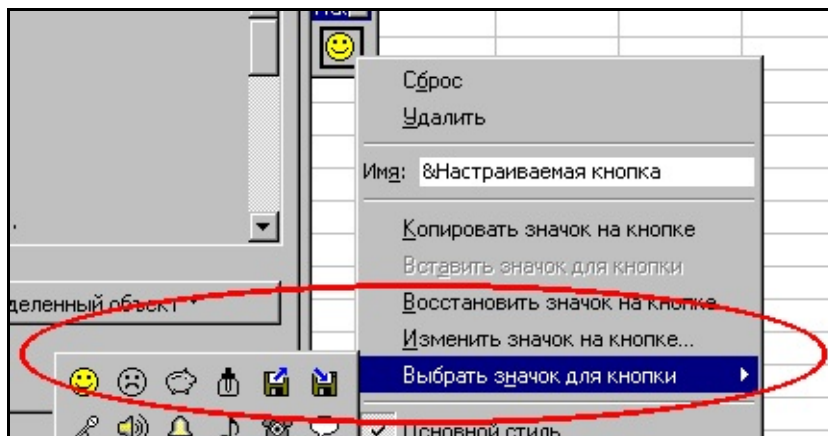




По нажатию на эту кнопку у Вас появится диалоговое окно с предложением ввести имя. После этого можно смело жать **ОК**. В Excel у Вас появится новая панель инструментов. Следующим шагом является помещение на эту панель настраиваемой кнопки. Нам нужно перейти на вкладку "Команды" и в списке "Категории" найти "Макросы", у Вас справа появится значек настраиваемая кнопка. Схватите ее мышкой и перенесите на панель инструментов.



С этого момента у нас есть возможность менять значек на кнопке. Вы можете выбрать готовый или создать свой. Нажмите правую кнопку мыши на кнопке и посмотрите.



После того как выбрали значек можно и привязать макрос. Опять по правой кнопке мыши есть пункт меню "Назначить макрос". Появится стандартное диалоговое окно с выбором макроса. Выберите его. Теперь осталось закрыть диалоговое окно. Если панель нужно отредактировать выберите опять этот пункт меню.

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 50 - Определяем выделенную ячейку

Давайте посмотрим как определить событие выбора определенной ячейки. Первое - это надо определить, что вообще что-то выбрали. Для этого создайте книгу. Запустите редактор **VBA** и щелкните на листе. Для данного листа будет создана функция **SelectionChange**, если ее нет, то у Вас всегда есть возможность ее выбрать. Теперь впишите код сообщения в функцию, чтобы увидеть, что происходит.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    MsgBox Target.Address
End Sub
```

У Вас теперь есть возможность щелкать по ячейкам конкретного листа, я еще раз повторяю и вы получите сообщение с адресом ячейки. Сообщение выбора ячейки обрабатывается локально. А что если у нас надо обрабатывать это сообщение с трех листов и желательно вместе. Для этого нужно создать модуль и в нем процедуру обработки. Я пока поместил туда просто вывод информационного окна.

```
Public Sub Selection_Cell(ByVal Address As String)
    MsgBox Address + " " + List
End Sub
```

И соответственно надо пересылать туда данный с каждого листа.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Selection_Cell Target.Address, Target
End Sub
```

Вот с этого момента можно вставлять вызов функции **Selection\_Cell** на каждый лист в ответ на реакцию выделения.

Но это еще не все. Дело в том, что передается объект типа **RANGE**, то есть

нет разницы передана одна ячейка или диапазон. Но при выделении диапазона внутри него может попасть и наша ячейка. Итак, нам надо бы анализировать диапазон. Давайте договоримся, что нам надо обязательно отловить момент выделения ячейки **\$A\$1**. Вот на всех листах. У нас для этого все есть. Процедура общая, в которой есть адрес. И реакция на выделение на каждом листе. Смотрим код:

```
Public Sub Selection_Cell(ByVal Address As String)
    On Error GoTo Ends
    Dim Test As Range
    Dim Find As Range
    Dim Result As Range
    Dim x As Integer

    Set Find = Range("$A$1")
    Set Test = Range(Address)

    Set Result = Intersect(Test, Find)
    x = Result.Count
    MsgBox "$A$1"

Ends:
End Sub
```

Идея простая. Мы имеем переданный диапазон **Test** и нужный **Find**. Как узнать, что один включает другой ??? Вызовем операцию пересечения **Intersect**. Результатом будет диапазон, который содержит пересечение. Так вот если искомая нам ячейка в нем есть, то нормально, а если нет, то обращение к **Count** вызовет ошибку. Этим мы и воспользовались.

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 51 - Изучаем события Excel Workbook

Открытие книги:

```
Private Sub Workbook_Open()  
End Sub
```

Активация окна. Вызывается в момент активации книги. Этот момент наступает в тот момент, когда из другой активной книги вы переходите к той, в которой обрабатывается событие. Например, при переключении книги в меню "Окно".

```
Private Sub Workbook_Activate()  
End Sub
```

Закрытие окна. Вызывается перед закрытием книги:

```
Private Sub Workbook_BeforeClose(Cancel As  
End Sub
```

Деактивация книги. Вызывается в момент перехода к другой книге:

```
Private Sub Workbook_Deactivate()  
End Sub
```

Активация листа. Вызывается в момент смены листа:

```
Private Sub Workbook_SheetActivate(ByVal S  
End Sub
```

Деактивация листа. Вызывается в момент смены листа для того листа, с которого уходят:

```
Private Sub Workbook_SheetDeactivate(ByVal  
End Sub
```

Смена ячейки. Вызывается в момент смены диапазона или при редактировании ячейки:

```
Private Sub Workbook_SheetChange(ByVal Sh  
End Sub
```

Пересчет данных. Вызывается при пересчете данных, обычно пересчет связан с редактированием данных, поэтому ранее вызывается **SheetChange** практически всегда за исключением ручного пересчета.

```
Private Sub Workbook_SheetCalculate(ByVal  
End Sub
```

Переход к ячейке. Вызывается при переходе от одной ячейки к другой:

```
Private Sub Workbook_SheetSelectionChange(  
End Sub
```

Разрешение на печать. Вызывается при выборе меню "Печать":

```
Private Sub Workbook_BeforePrint(Cancel As  
End Sub
```

Разрешение на сохранение. Вызывается перед сохранением документа:

```
Private Sub Workbook_BeforeSave(ByVal Save  
End Sub
```

Создание нового листа. Вызывается в момент создания нового листа. Обычно ведет за собой цепочку событий - **Workbook\_NewSheet**, **Workbook\_SheetDeactivate**, **Workbook\_SheetActivate**:

```
Private Sub Workbook_NewSheet(ByVal Sh As Worksheet)
End Sub
```

Изменение размера листа. Вызывается в момент изменения размера листа:

```
Private Sub Workbook_WindowResize(ByVal Wn As Worksheet)
End Sub
```

Активизация окна. Вызывается в момент активизации окна книги, например, при переключении на книгу:

```
Private Sub Workbook_WindowActivate(ByVal Wn As Worksheet)
End Sub
```

Щелчок правой кнопкой. Вызывается по нажатию правой кнопки мыши:

```
Private Sub Workbook_SheetBeforeRightClick
End Sub
```

Выводы. Часть событий связана в цепочки. То есть, например, событие **Activate** для одного объекта это событие **Deactivate** для другого объекта. Это работает для Окон, Книг и Листов. Часть событий вызывают за собой следующие события, например, ввод данных приведет к пересчету листа. Или вставка нового листа к событиям активизации. Знание событий может помочь в решении нетривиальных вопросов. Вот так можно запретить печать книги.

```
Private Sub Workbook_BeforePrint(Cancel As Integer)
Cancel = True
End Sub
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 52 - Автоматизация на основе СУММЕСЛИ

**Excel** предоставляет огромное количество возможностей и знание этих возможностей значительно может помочь при создании приложений. Давайте рассмотрим обычную ситуацию с созданием счетов. У вас есть список товаров с ценами. Вы готовите счет с набором товаров в разных количествах.

Создаем новую книгу и лист переименовываем в "Цены", а на нем создаем два именованных диапазона. Диапазон **name** и диапазон **price**. В одном будет название товара, во втором цена. Внимание **название товаров должно быть уникальным**. То есть в таком варианте если делать как описано в шаге. Конечно сейчас на складах есть одни и те же товары с разными ценами. Но ведь к названию можно что-то добавить ??? Правда же ??? Именовывать нужно целую колонку. Зачем ??? Чтобы спокойно добавлять новые товары и не думать о изменениях. Вот посмотрите рисунок демонстрирующий суть идеи.

	A	B	C	D	E
1		УАЗ 469	100		
2		УАЗ 469 В	200		
3		МАЗ	300		
4		КРАЗ	200		
5					
6					
7					
8					

Делаем второй лист с названием "продаем". Здесь мы будем формировать наш счет. Помножить количество на цену и получить сумму, ну это не проблема. А вот как узнать цену исходя из названия ??? Тут нам и поможет **СУММЕСЛИ**, вот смотрите:



A	B	C	D	E	F
	Товар	Количество	Цена	Сумма	
	УАЗ 469	1	100	100	

Здесь написано, что суммировать по диапазону **price**, если в диапазоне **name** попадет название равное указанному в ячейке **B4**. На самом деле если названия уникальны никакого суммирования не будет, просто выберется нужная цена. Почему диапазон сделан на целую колонку ??? Чтобы спокойно добавлять или удалять названия и ни о чем не думать. И еще если в диапазоне указать что-то типа **A1:A100**, то при растаскивании вниз **Excel** будет менять адреса. А вот если у Вас диапазон не на всю колонку, то при растаскивании формул диапазон не будет меняться.

Ну вот и все. Достаточно ввести название, количество и вы получите общую цену. Надо еще один товар растащите формулу вниз и она будет работать. Теперь при доработке листа вам останется лишь вводить названия и количество.

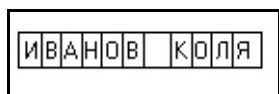
B	C	D	E
Товар	Количество	Цена	Сумма
УАЗ 469	1	100	100
		0	0
		0	0
		0	0
		0	0
		0	0
		0	0
		Итого по счету	100

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 53 - Разбор строки стандартными функциями

Итак, ситуация простая. У нас в одной ячейке записано имя и фамилия человека. Конечно это не хорошо. Но дело сделано и так оно есть. Строка представляет из себя набор символов, даже если она в ячейке **Excel** :-)



Соответственно раз это набор символов, то должны быть и функции для определения количества символов. Они есть. Вот она **=ДЛСТР(B5)**, если в **B5** будет та надпись, то она вернет 11. Естественно, что первая буква будет первой :-)

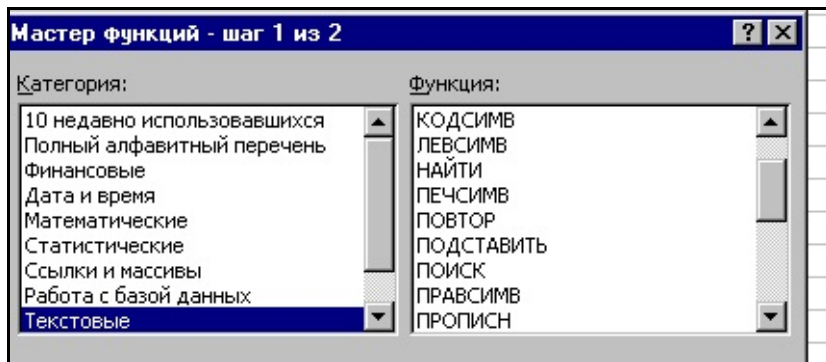
Чтобы разделить имя и фамилию нам надо найти символ, на котором заканчивается имя. Этим символом почти всегда является пробел. Должна быть функция, которая умеет искать место символа в строке. Она тоже есть, **=НАЙТИ(" ";B5;1)** вернет первую позицию, в которой находится пробел. В данном случае 7.

Для отбора строки нам надо выбрать символ от начала до пробела и от пробела до конца. Начало мы знаем - это 1, пробел знаем - это 7, конец тоже - это 11. Осталось найти только функцию вырезания.

**=ПСТР(B5;1;D5)**

**=ПСТР(B5;D5;C5)**

Эта функция вырезает из строки символы от указанного значения до указанного. Соответственно от начало до пробела - это фамилия, а от пробела до начала - имя. Все функции есть в мастере "функция" в разделе "текстовые".



Вывод отсюда простой. Многие задачи можно сделать и без функций работы со строками **VB(A)**. И, наверно, разумный компромис между **VBA** и возможностями **Excel** - это и есть мастерство создания приложений на базе **Excel**.

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 54 - Подробнее о событиях загрузки и выгрузки формы

Форма в **VBA** это каркас приложения. Как добавлять форму в Ваш проект смотрите "[Шаг 15 - Пользовательские формы](#)". В основном события формы по ее инициализации и деинициализации разворачиваются в таком порядке:

Initialize  
Load  
Activate  
Deactivate  
QueryUnload  
Unload  
Terminate

Но форма в **VBA** и **VB** различаются. Давайте сравним:

VBA	VB
UserForm_Initialize()	Form_Initialize()
Нет	Form_Load()
UserForm_Activate()	Form_Activate()
UserForm_Deactivate()	Form_Deactivate()
UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)	Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
нет	Form_Unload(Cancel As Integer)
UserForm_Terminate()	Form_Terminate()

Получается, что событие **Load** и **Unload** в **VBA** не обрабатываются.

### Событие Initialize

Событие **Initialize** (Инициализация) обычно используется для подготовки к работе приложения или формы **UserForm**. Переменным присваиваются исходные значения, а положение или размеры элементов управления могут быть изменены для согласования с данными, заданными при инициализации. Это событие появляется до загрузки формы и ее отображения. Это событие появляется во время загрузки формы. Давайте в него напишем код:

```
Private Sub UserForm_Initialize()  
    MsgBox "UserForm_Initialize"  
End Sub
```

А теперь две функции, которые вызывают **Load**:

```
Sub Test()  
    Load UserForm1  
    Call Test2  
End Sub  
  
Sub Test2()  
    Unload UserForm1  
    Load UserForm1  
End Sub
```

В результате окно с информацией о инициализации будет на экране два раза. Так же это событие сгенерирует событие **Show**, так как первый раз для работы с формой ее нужно загрузить. Еще это событие может быть вызвано, если в форме определена функция общего назначения. Вызов этой функции опять приводит к загрузке формы.

```
Private Sub UserForm_Terminate()  
  
End Sub  
  
Public Sub MyMessage()
```

```
        MsgBox "MyMessage"  
    End Sub
```

А вот так можно вызвать:

```
Sub Test()  
    UserForm1.MyMessage  
End Sub
```

Итак, событие **Initialize** вызывается только один раз для формы перед ее загрузкой.

## Событие Load

Нет ее в **VBA**, а вообще в **VB** здесь можно что-то сделать перед выводом формы на экран.

## Событие Activate и Deactivate

Событие **Activate** происходит, когда объект становится активным окном. А становится активным окном он может в двух случаях. Это в результате **Show**, когда форма становится видимой на экране и в результате получения фокуса. Событие **Deactivate** (Деактивизация) происходит, когда объект более не является активным окном. Эти события генерируются только при переключении между окнами одного приложения. Если вы перейдете в другую программу и вернетесь в **Excel**, то эти события не будут сгенерированы.

## Событие QueryClose

Это событие генерируется для того, чтобы получить у пользователя разрешение на закрытие формы. В зависимости от кода возврата форма

закрывается или нет.

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    If CloseMode <> 1 Then Cancel = 1
    UserForm1.Caption = "Close"
End IF
End Sub
```

## Событие Unload

Нет его, точно так же как и **Load**, а вообще используется, если надо что-либо делать при выгрузке формы, например сохранить настройки пользователя.

## Событие Terminate

Данное событие происходит, когда все ссылки на экземпляр объекта удаляются из памяти с помощью присвоения всем переменным, которые ссылаются на данный объект, состояния **Nothing** или когда последняя ссылка на объект выходит за пределы области определения. Это событие идет вслед за **Unload**.

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 55 - Четыре основных метода работы с формой (Load,Unload,Show,Hide)

Давайте сформулируем краткие описания и после этого посмотрим как они взаимодействуют:

- **Load** - загрузка формы в память
- **Unload** - выгрузка формы из памяти
- **Show** - показ формы на экране
- **Hide** - спрятать форму

Мы помним, что **Show** может сам вызвать **Load**, но если форма загружена, то **Load** не будет вызываться. Поэтому просто надо делать так:

```
Начало работы
Load
Конец работы
Unload
```

Показывать или прятать форму можно с помощью **Show** и **Hide**. Но, если Вы проводите в методе **Initialize** настройку переменных, то будьте уверены, что она будет происходить при загрузке в момент вызова **Load**.

Вот типовые коды:

```
Load
Load UserForm

Unload
Unload UserForm
Из формы
Unload Me
```



Hide  
UserForm.Hide  
Из формы  
Me.Hide

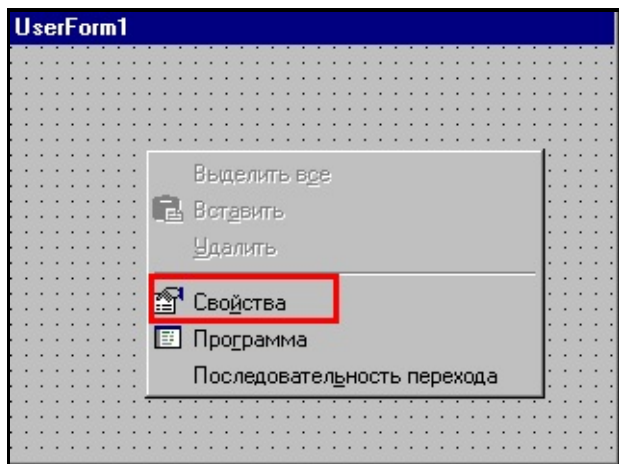
Show  
UserForm.Show

---

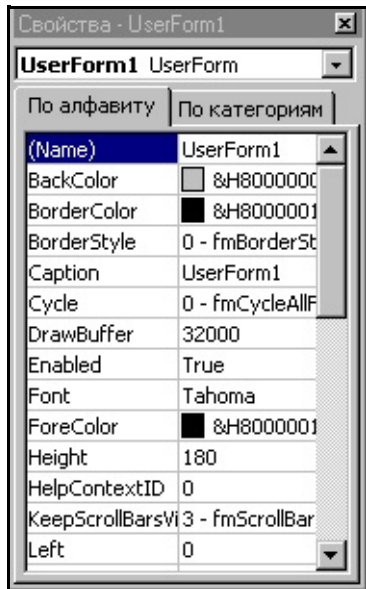
[Предыдущий Шаг](#) | [Следующий Шаг](#)  
Автор [Каев Артем](#).

## Шаг 56 - Настройка свойств формы

Свойства формы можно менять, для этого достаточно вызвать меню левой кнопкой на форме.



В результате этого появится окно свойств формы.



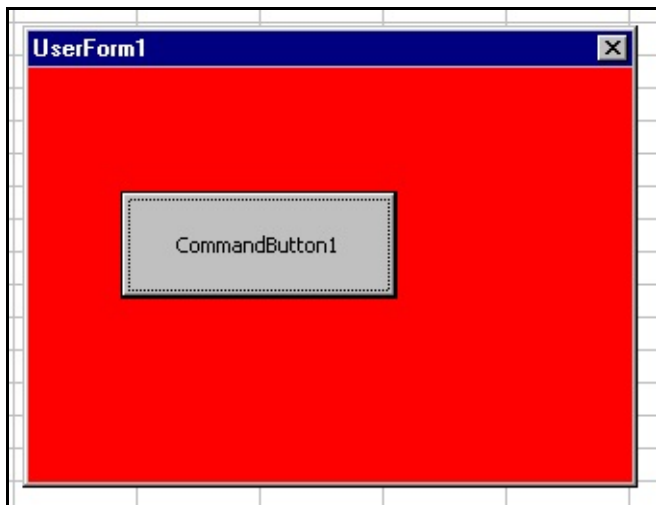
Так же свойства формы можно изменять и в период выполнения. Для этого нужно указать форму, затем через точку название свойства и использовать присваивание для смены свойства. Давайте, например, поменяем цвет формы по нажатию на кнопку на самой же форме.

```
Private Sub CommandButton1_Click()  
    UserForm1.BackColor = RGB(255, 10,  
End Sub
```

Если запустить эту форму по нажатию на кнопку, то она станет красная.

```
Sub test()  
    Load UserForm1  
    UserForm1.Show  
    Unload UserForm1  
End Sub
```

А вот результат.



---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

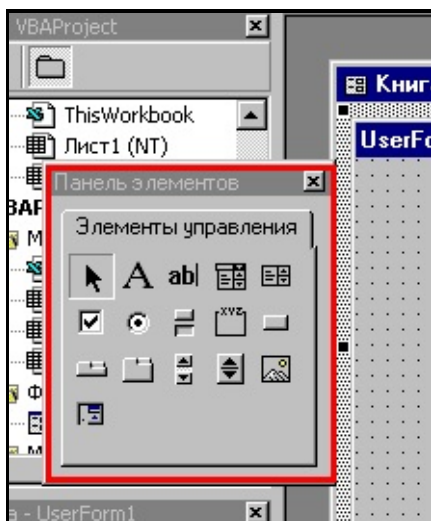
Автор [Каев Артем](#).

## Шаг 57 - Элементы для формы

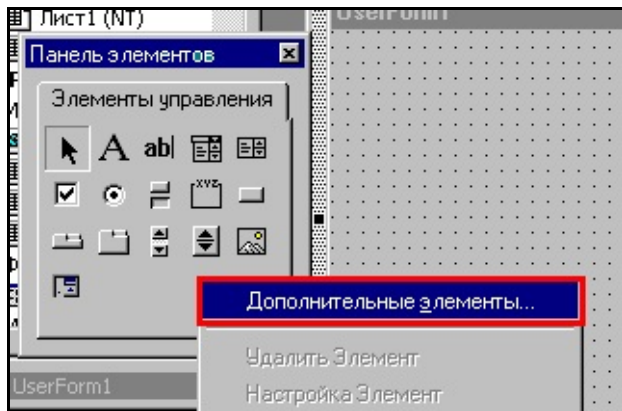
Элемент управления, наверно, вторая главная составляющая интерфейса **VBA** после форм. Элементы управления позволяют Вам добавить в программу функциональность. Их два типа:

- Встроенные - **inherent**
- Нестандартные - **custom**

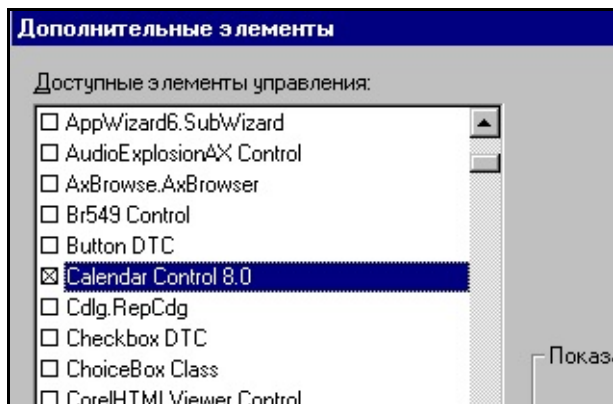
По умолчанию в новом проекте в панели инструментов находятся только нестандартные элементы управления.



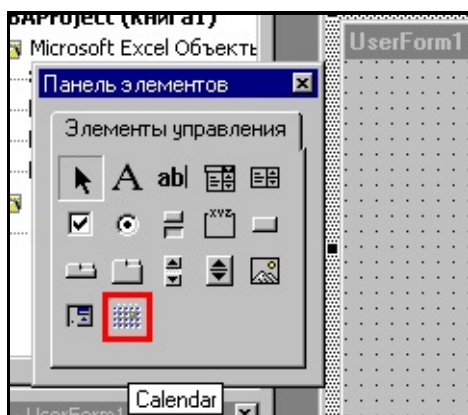
Но у Вас есть возможность добавлять в эту панель элементы **ActiveX**, которые могут существенно увеличить функциональность. Эти элементы в виде файлов **OCX** на данный момент реализовывают практически все. Достаточно нажать правую кнопку мыши на свободном месте в форме элементов управления и появится меню.



Щелкнув по меню "дополнительные элементы" вы сможете выбрать любой элемент из тех, которые установлены в системе. Например, выберите календарь и нажмите кнопку **ОК**.

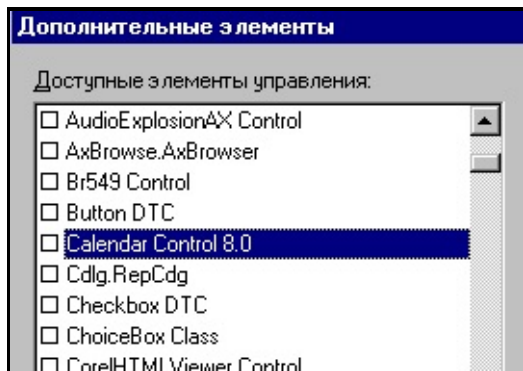


После этого значок будет в элементах управления.



Отсюда его можно помещать на форму. Для удаления его с панели инструментов достаточно снять галочку напротив него в дополнительных

КОМПОНЕНТАХ.



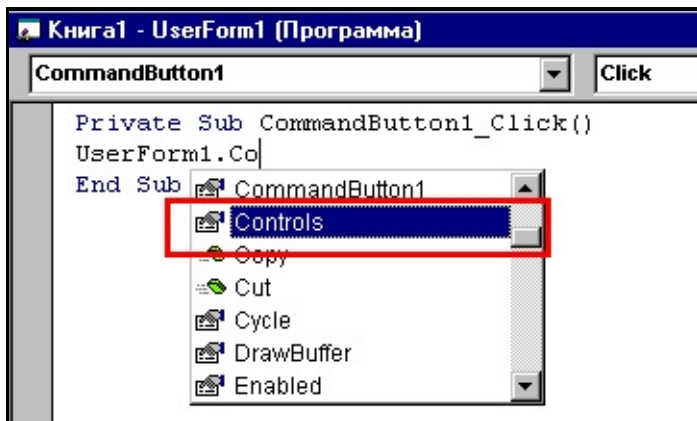
---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 58 - Наборы элементов управления

В шаге ["Шаг 46 - Доступ к одинаковым элементам управления"](#) я показывал код как можно пройтись по всем элементам управления, но ничего практически не рассказал. Давайте обсудим это подробнее. **VBA** предоставляет массив включающий все элементы на форме с именем массива **Controls**:



У этого массива есть ряд методов, но только одно свойство. Это свойство **Count**. Данное свойство возвращает количество элементов на форме.

```
Private Sub CommandButton1_Click()  
    MsgBox UserForm1.Controls.Count  
End Sub
```

С помощью этого массива можно ссылаться на элемент по индексу или по имени. У меня на форме два текстовых элемента, вот я их и спрячу двумя способами по нажатию на кнопку.

```
Private Sub CommandButton1_Click()  
    Controls(0).Visible = False  
    Controls("TextBox2").Visible = False  
End Sub
```

Для того, чтобы пробежаться по всем элементам можно использовать цикл

**For Each.** Следующий код скрывает все текстовые элементы управления:

```
Private Sub CommandButton1_Click()  
    Dim ctrl As Control  
    For Each ctrl In Controls  
        If TypeName(ctrl) = "TextB  
            ctrl.Visible = Fal  
        End If  
    Next ctrl  
End Sub
```

Функция **TypeName** возвращает значение типа **String**, представляющее тип переменной за исключением типа определенного пользователем с помощью **Type**.

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).



## Шаг 59 - Проверка ввода на уровне формы (KeyDown,KeyUp,KeyPress)

События:

```
KeyDown  
KeyUp  
KeyPress
```

Посылаются форме когда производится ввод данных. Событие **KeyPress** возникает когда пользователь нажимает клавишу, у которой есть **ASCII** код. Это событие не сгенерируется при нажатии функциональных клавиш. При том эти события есть как у формы:

```
Private Sub UserForm_KeyPress(ByVal KeyAsc.  
  
End Sub
```

Так и у элементов управления **TextBox**, например:

```
Private Sub TextBox1_KeyDown(ByVal KeyCode  
  
End Sub
```

А это в свою очередь позволяет проверять ввод. Например, если в поле нужно ввести цифру, то легко проверить, что вводит пользователь. События **KeyUp** и **KeyDown** применяются для специальных функциональных клавиш или комбинаций типа **Ctrl+Del**, так как событие **KeyPress** их не отловит. Удобно использовать данное событие для проверки заполненности полей. Вот форма:

А вот код для нее. В момент нажатия ввода символа в поле проверяется заполненность полей:

```
Private Sub Test()  
    Dim ctrl As Control  
    Dim bool As Boolean  
    bool = True  
    For Each ctrl In Controls  
        If TypeName(ctrl) = "TextB  
            If ctrl.Text = ""  
                bool = Fal  
            End If  
        End If  
    Next ctrl  
    UserForm1.CommandButton1.Enabled =  
End Sub  
  
Private Sub TextBox1_KeyPress(ByVal KeyAsc.  
    Call Test  
End Sub  
  
Private Sub TextBox2_KeyPress(ByVal KeyAsc.  
    Call Test  
End Sub  
  
Private Sub TextBox3_KeyPress(ByVal KeyAsc.  
    Call Test  
End Sub
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

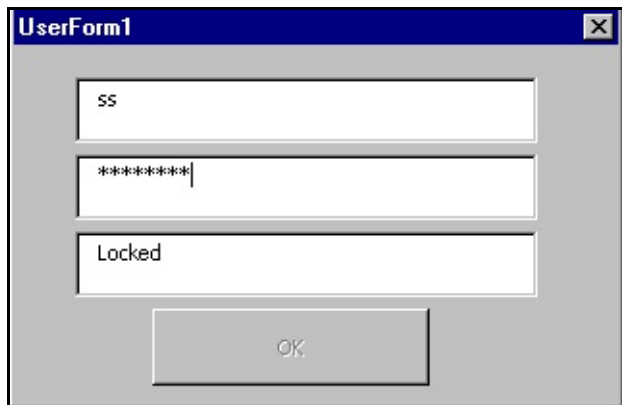
Автор [Каев Артем](#).

## Шаг 60 - Проверка и настройка ввода в TextBox

У элемента управления есть несколько свойств, которые позволяют на этапе проектирования ограничить ввод пользователя. Вот они:

MaxLength  
PasswordChar  
Locked

Свойство **MaxLength** позволяет ограничить количество символов, которые будут введены. Если пользователь попытает ввести больше, то будет звуковой сигнал. **PasswordChar** не дает возможности просматривать вводимые символы заменяя их на звездочки (\*). Это полезно, как видно из имени, при вводе пароля. Свойство **Locked** определяет может ли пользователь редактировать поле. Посмотрите пример:



В верхнем поле можно ввести только два символа. Во втором поле вместо символов звездочки, а последнее поле нельзя изменить. Кстати в **PasswordChar** не обязательно должна быть звездочка, то есть может быть любой другой знак.



Но в большинстве приложений принята звездочка и смысла пугать пользователя экзотическими знаками наверно нет.

Кстати **Locked** говорит, что поле для чтения, но в некоторых ситуациях его можно открыть для редактирования программным путем.

```
Private Sub CommandButton1_Click()  
    TextBox3.Locked = False  
End Sub
```

Тоже самое и с **MaxLength**:

```
Private Sub CommandButton1_Click()  
    TextBox3.Locked = False  
End Sub
```

И **PasswordChar**:

```
Private Sub CommandButton1_Click()  
    TextBox2.PasswordChar = "x"  
End Sub
```

При изменении свойств данные в полях не пропадают.

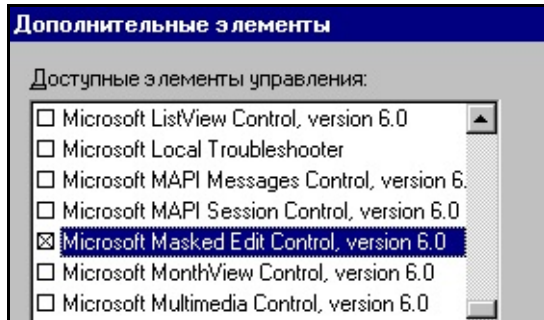
---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

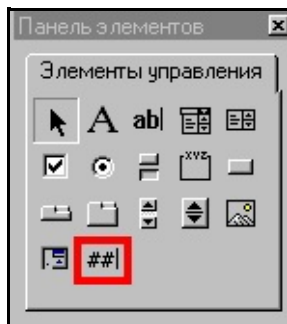
Автор [Каев Артем](#).

## Шаг 61 - О MaskedTextBox

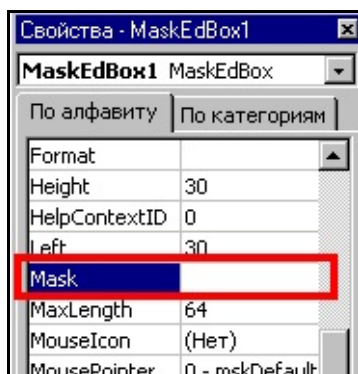
Это элемент ввода, в котором можно использовать маску ввода. Вам нужно ее добавить.



В результате на панели инструментов добавится новый значок.



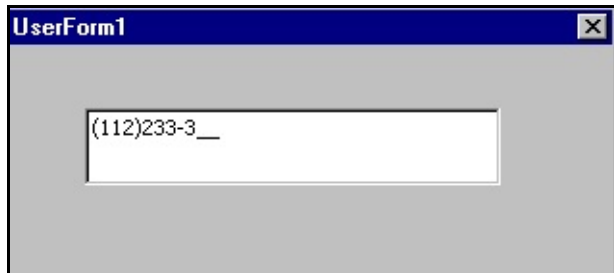
Одно из его важных свойств - это свойство **Mask**:



С помощью этой маски легко задать форматирование текста. Например, вот такое для ввода телефона.

(###)###-###

Теперь при вводе у Вас будет шаблон:



Для программной очистки необходимо ввести саму маску, например, вот так:

```
Private Sub CommandButton1_Click()  
    MaskedTextBox1.Text = "(____)____-____"  
End Sub
```

Кодов для ограничения ввода много, но вот минимальные:

# - ЧИСЛО  
a - СИМВОЛ

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 62 - Maskedit - Text и ClipText

Эти два свойства позволяют получить данные из элемента **Maskedit**. Отличие этих свойств в том, что поле **Text** возвращает строку вместе с маской, а **ClipText** без маски. Давайте посмотрим. Вот данные введенные в маску.



Давайте посмотрим как они будут возвращаться в случае применения этих двух свойств. Делаем код для кнопок:

```
Private Sub CommandButton1_Click()  
    Debug.Print MaskedTextBox1.Text  
    Debug.Print MaskedTextBox1.ClipText  
End Sub
```

И смотрим результат:

```
(12)34  
1234
```

Данные свойства удобны при помещении данных в базу данных и обратно с маской, которую база данных не поддерживает.

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).



## Шаг 63 - Обработка ошибок в VBA

Обрабатывать ошибки можно тремя способами:

1. Строчная обработка
2. Создание локального обработчика
3. Создание глобального обработчика

Для строчной обработки ошибок применяется функция **On Error Resume Next**, при использовании этой функции выполнение работы программы не прерывается. В этот момент в объект **Err** помещается код ошибки, который можно выяснить через свойство **Number**. После обработки ошибки его необходимо очистить, воспользовавшись методом **clear**:

```
Sub Test()  
    On Error Resume Next  
    Open "c:\nullfile.nul" For Input A  
    Select Case Err.Number  
    Case 53:  
        MsgBox "Not file"  
    Case 55:  
        MsgBox "Not access"  
    End Select  
    Err.Clear  
End Sub
```

Строчный обработчик можно отключить.

```
On Error Goto 0
```

Локальный обработчик специфичен для конкретной процедуры. То есть для каждой процедуры вы создаете свой обработчик ошибок. Общий вид такой:

```
On Error Goto ErrorHandler
    КОД
Exit Sub
ErrorHandler:
    Код обработки ошибки
End Sub
```

Обратите внимание на **Exit Sub**, который предназначен для выхода из процедуры. Если этого не сделать, то код предназначенный для обработки ошибок все равно выполнится, а это недопустимо. После обработки ошибки вы должны возобновить работу программы. Есть три способа:

- Повтор выполнения строки - **Resume**
- Выполнение следующей строки - **Resume Next**
- Закрытие формы - **Unload Me**

При отсутствии обработчика ошибок будет произведен поиск обработчика в вызывающей процедуре, если там его нет, то дальше по цепочке вызовов. Но при этом возобновление выполнения команд будет довольно сложным. Так как **Resume**, **Resume Next** будет работать в зависимости от того в какой процедуре оказалась ошибка.

```
Sub ErrorTest()
    On Error GoTo Error:
    Call Test
Error:
    Select Case Err.Number
    Case 53:
        MsgBox "Not file"
    Case 55:
        MsgBox "Not access"
    End Select
    Err.Clear
```

```
End Sub
```

```
Sub Test()
```

```
    Open "c:\nullfile.nul" For Input A
```

```
End Sub
```

Централизованная обработка ошибок необходима, если ошибки могут возникнуть в разных местах, а обрабатывать лучше в одном. Например, во многих местах программы производится создание файлов. Идея заключается в том, что создание файла всегда производится в одной процедуре где и проводится обработка ошибок.

```
Sub ErrorTest()
```

```
    Call Test
```

```
End Sub
```

```
Sub Test()
```

```
    On Error GoTo Error:
```

```
    Open "c:\nullfile.nul" For Output ,
```

```
    Close #1
```

```
    Exit Sub
```

```
Error:
```

```
    MsgBox "error"
```

```
    Err.Clear
```

```
End Sub
```

Так вот теперь процедуру **Test** можно вызывать из разных мест и всегда будет произведена одна и та же обработка ошибок. Таким образом удобнее поддерживать процедуры обработки, так как они централизованные.

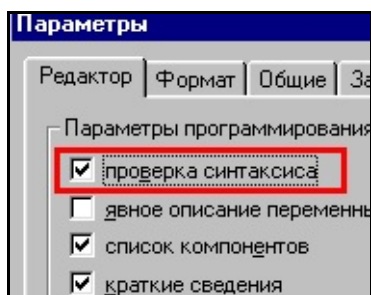
---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

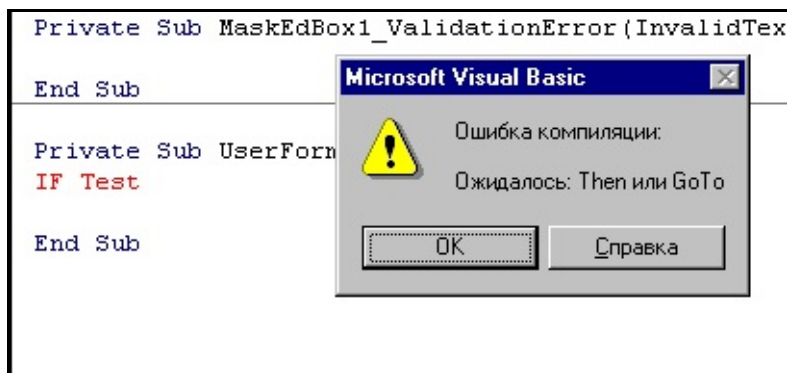
Автор [Каев Артем](#).

## Шаг 64 - Функция автоматической проверки синтаксиса

Как в **VB**, так и в **VBA** автоматически проверяются синтаксические ошибки, но только в том случае, если проверка включена в **Сервис -> Параметры**.



Если эта опция включена, то при проверке синтаксиса Вам будет выводиться сообщение о том, что Вы неправильно пишете. Например, если не завершить **IF**, то сразу после набора строки Вы получите сообщение.



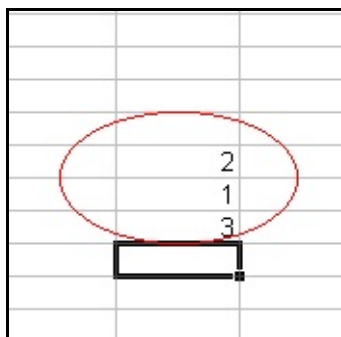
---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 65 - Выделение диапазона выше текущей ячейки

Задача выделить диапазон выше текущей ячейки:



А вот и код с комментариями:

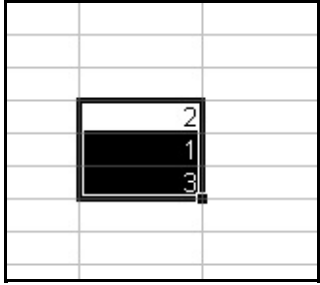
```
Sub SelectColumnData()  
    ' что делать при ошибке  
    On Error GoTo errors  
    ' нижний адрес  
    Dim a1 As String  
    ' верхний адрес  
    Dim a2 As String  
    ' диапазон  
    Dim ran As Range  
    ' если не верхняя ячейка  
    If (ActiveCell.Row > 1) Then  
        ' пойти вверх  
        ActiveCell.Offset(-1, 0).Select  
        ' взять адрес ячейки  
        a1 = ActiveCell.Address  
        ' будем подниматься  
        For x = 1 To (ActiveCell.Row - 1)  
            ' на одну вверх
```

```

        ActiveCell.Offset(
            ' если не число вы
        If IsNumeric(Active
            ' на одну
            ActiveCell
            ' ВЫХОД
            GoTo nexts
        End If
        ' если пустая
        If IsEmpty(ActiveC
            ' на одну
            ActiveCell
            ' ВЫХОД
            GoTo nexts
        End If
    Next x
nexts:
    ' получаем адрес верхней
    a2 = ActiveCell.Address
    ' строим диапазон
    Set ran = Range(a1 + ":" +
    ' выделяем
    ran.Select
End If
' выходим из процедуры
Exit Sub
' ошибка, зовем на помощь
errors:
MsgBox "Ошибка сообщите разработчи
End Sub

```

А вот и результат:



---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 66 - Движение по диапазону

Выделив диапазон может возникнуть задача пройтись по этому диапазону с целью, например, покраски значений определенного диапазона. Смотрим:

```
Sub FullShach()  
    For Each c In Range(addressdiap)  
        If c.Value > yr1 Then  
            c.Select  
            With Selection.Interior  
                .ColorIndex = 1  
                .Pattern = 1  
            End With  
            Selection.Font.Color = c.Value  
            If c.Value > yr2 Then  
                c.Select  
                Selection.Interior.ColorIndex = c.Value  
                Selection.Interior.Pattern = 1  
            End If  
        End If  
    Next c  
End Sub
```

Основа этой функции цикл **For Each c In Range(addressdiap)**, который будет перебирать ячейки в диапазоне и возвращать каждую ячейку в переменную цикла **c**. Для того, чтобы можно было произвести работу с этой ячейкой ее надо выделить **c.Select**.



Автор [Каев Артем](#).

## Шаг 67 - Движение по ячейкам

Для движения по таблице используется функция.

```
переменная.Offset(RowOffset, ColumnOffset)
```

В качестве переменных может выступать как ячейка так и диапазон (**Range**) удобно пользоваться этой функцией для смещения относительно текущей ячейки.

Например, смещение вниз на одну ячейку и выделение ее:

```
ActiveCell.Offset(1, 0).Select
```

Если нужно двигаться вверх, то нужно использовать отрицательное число:

```
ActiveCell.Offset(-1, 0).Select
```

Функция ниже использует эту возможность для того, чтобы пробежаться вниз до первой пустой ячейки.

```
Sub beg()  
    Dim a As Boolean  
    Dim d As Double  
    Dim c As Range  
    a = True  
    Set c = Range(ActiveCell.address)  
    c.Select  
    d = c.Value  
    c.Value = d  
    While (a = True)  
        ActiveCell.Offset(1, 0).Se.  
        If (IsEmpty(ActiveCell.Val  
            Set c = Range(Acti
```

```
                c.Select
                d = c.Value
                c.Value = d
            Else
                a = False
            End If
        Wend
    End Sub
```

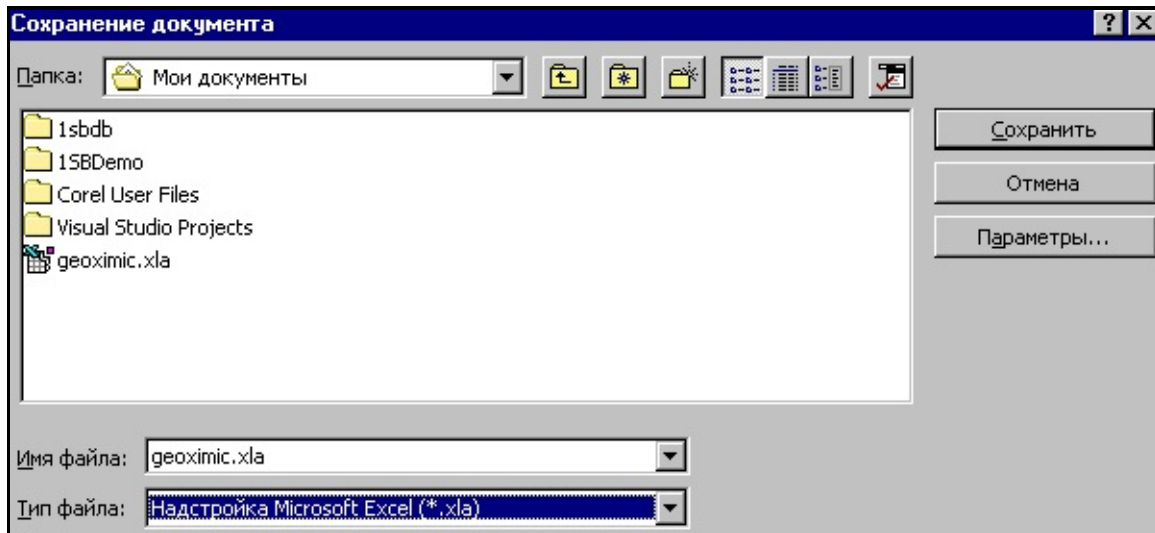
---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 68 - Как сделать XLA ?

Дополнения к **Excel** делаются на основе обычной книги. Нужно создать книгу, потом написать макросы. После этого сохранить книгу как **XLA**.



В результате у нас будет файл с расширением **XLA**.



В книге есть несколько событий, которые можно использовать для настройки меню или любых других действий при установке дополнения. Вот они:

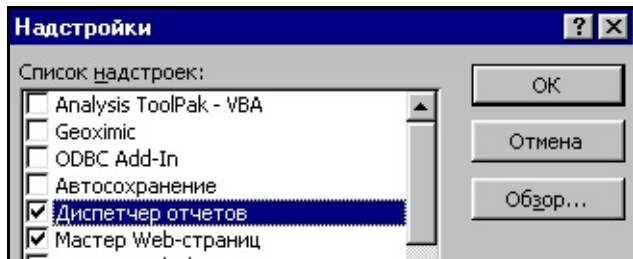
```
Private Sub Workbook_AddinInstall()
```

```
End Sub
```

```
Private Sub Workbook_AddinUninstall()
```

```
End Sub
```

Они будут вызываться при установке или удалении дополнения через меню "сервис -> надстройки".



---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 69 - Динамическое создание меню

Смотрим код:

```
Const menuname = "Геохимия"

Private Sub Workbook_AddinInstall()
    ' обработка ошибки
    On Error GoTo errors:
    Dim num As Integer
    ' получаем количество пунктов меню
    num = Application.CommandBars("Wo
    ' добавляем 1 для следующего
    num = num + 1
    Dim a As CommandBarControl
    ' создаем новый пункт меню
    Set a = Application.CommandBars("W
    ' даем имя
    a.Caption = menuname
    Dim help As CommandBarControl
    Set help = Application.CommandBars
    help.Caption = "Помощь"
    help.OnAction = "Help"
    Dim comms As CommandBarControl
    Set comms = Application.CommandBar
    comms.Caption = "Расчет аномальных
    comms.OnAction = "Anomal"
    Dim commslog As CommandBarControl
    Set commslog = Application.Command
    commslog.Caption = "Расчет аномаль
    commslog.OnAction = "Anomallog"
    Dim commsbeg As CommandBarControl
```

```

Set commsbeg = Application.CommandBars.Add("commsbeg")
commsbeg.Caption = "Пробежаться по комментам"
commsbeg.OnAction = "beg"
Exit Sub
' что будем делать при ошибке
errors:
MsgBox Err.Description + " сообщите об ошибке"
Err.Clear
End Sub

```

Основа кода коллекция **CommandBars**, которая отвечает коллекции меню. У этой коллекции есть метод **Add**, после которого меню надо присвоить название и имя макроса.

```

help.Caption = "Помощь"
help.OnAction = "Help"

```

Удаляется меню по имени. Вот код:

```

Private Sub Workbook_AddinUninstall()
On Error GoTo errors:
Application.CommandBars("WorksheetFunction:SUM")
Exit Sub
' что будем делать при ошибке
MsgBox "не могу удалить пункт меню"
errors:
End Sub

```

Если Вы обратите внимание, то меню устанавливается при подключении расширения **Excel**, а удаляются при его отключении.

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 70 - Нефть, таблицы и как делать не надо

Вот такое письмо.

Добрый день Артем.

Дело в том что с этими данными также нужно другие расчеты и представлять их геологам, моему это сделать будет немного проблематично для передачи. так что лучше наверное все же понять в чем именно суть проблемы да? еще раз уж не было непоняток, а то может объяснил :  
1 столбец имя скважины, второй насыщение п  
Выглядит это следующим образом:

1210k	Нефть
1210k	Нефть
1210k	НВ
1210k	Неясно
1231	Вода
1231	Вода
1231	Вода
1231	Вода

По скважине 1210k есть насыщение нефть, нв скважины нефтенасыщенный 1231 только вода : в таком же духе. Вот.

С уважением Рустам Сафиуллин  
[mailto: rustam@geodata.ru](mailto:rustam@geodata.ru)

Первая мысль которая родилась у меня после этого письма послать все к черту вместе с автором. Это классическая задача баз данных. Любые расчеты и все такое можно решить с помощью того же **ACCESS** и намного проще. Кроме того большая часть кода ниже будет просто реализация



стандартного SQL запроса. Кроме того, код подвержен ошибкам в данных ведь Нефт и Нефть не одно и тоже. Но мой опыт работы с геологами показывает что объяснять им что то бесполезно. Можно и на EXCEL сделать. Можно неправильно но можно. Итак таблица выгляди вот так.

	А	В
1	1210k	Нефть
2	1210k	Нефть
3	1210k	НВ
4	1210k	Неясно
5	1231	Вода
6	1231	Вода
7	1231	Вода
8	1231	Вода

Сначала нужно определить где начинаются и заканчиваться данные, это написано в ["Шаг 45 - Начало и конец данных"](#).

```
Dim allbore As Range ' здесь будет хранить  
' выбрать колонку  
Set allbore = Range("A:A")  
' только с данными  
Set allbore = Range(allbore.Columns.End(xl  
' выделить  
allbore.Select
```

А вот результат:

А	В
1210k	Нефть
1210k	Нефть
1210k	НВ
1210k	Неясно
1231	Вода
1231	Вода
1231	Вода
1231	Вода

Теперь нам нужно создать список скважин которые есть, способ один перебрать все записи и уникальные поместить у коллекцию. Как двигаться по диапазону написано в шаге ["Шаг 66 - Движение по диапазону"](#).

```

Dim borename As New Collection ' это набор

Sub FindOil()
    Dim allbore As Range ' здесь будет
    Set allbore = Range("A:A") ' выбра
    ' ТОЛЬКО С ДАННЫМИ
    Set allbore = Range(allbore.Column
    allbore.Select
    For Each bore In allbore
        bore.Select
        borename.Add (bore.Value)
    Next bore
End Sub

```

Вот теперь у нас в коллекции все имена скважин. Но они повторяются же. Надо при добавлении проверять есть такое имя в коллекции или нет. Напишем функцию.

```

Function FindElement(name As String) As Bo
    ' бежим по коллекции
    For Each elem In borename
        ' если имя совпадает верну
        If elem = name Then
            FindElement = Fals
            Exit Function
        End If
    Next elem
    ' нет имени
    FindElement = True
End Function

```

И применим ее:

```

Dim allbore As Range ' здесь будет хранить

```

```
Sub FindOil()  
    Set allbore = Range("A:A") ' выбра  
    ' ТОЛЬКО С ДАННЫМИ  
    Set allbore = Range(allbore.Column  
    allbore.Select ' выделить  
    For Each bore In allbore ' бежим по  
        bore.Select ' выделяем ячейку  
        If FindElement(bore.Value) = T  
            ' если скважины нет в колл  
                borename.Add (bore  
        End If  
    Next bore  
End Sub
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 71 - Нефть, таблицы и как делать не надо, продолжение

Итак, теперь нам нужно в соответствии с номером скважины делать выборку. Заведен еще один массив.

```
Dim allbore As Range           ' здесь бу
Dim alldata As Collection      ' это набо
Dim borename As New Collection ' это набо
```

Напишем функцию которая будет заполнять коллекцию по именам скважины:

```
Sub SelectBore(s As String)
    Set alldata = New Collection
    For Each bore In allbore ' бежим по
        bore.Select ' выделяем ячейку
        If bore.Value = s Then ' если совпадает
            ActiveCell.Offset(0,1).Text = bore.Value
            alldata.Add (ActiveCell)
        End If
    Next bore
End Sub
```

Пробежим по всем скважинам и возьмем значение из правой колонки поместив его в массив.

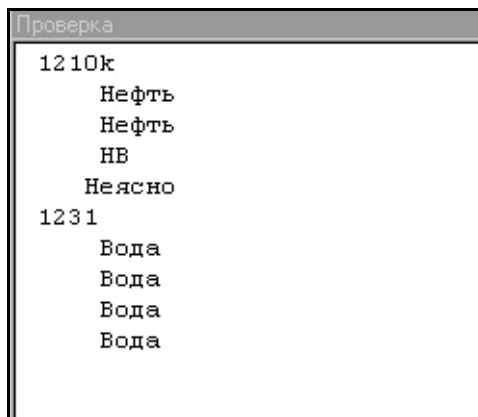
```
Sub FindOil()
    Set allbore = Range("A:A") ' выбрать
    ' ТОЛЬКО С ДАННЫМИ
    Set allbore = Range(allbore.ColumnNumber, allbore.ColumnNumber, allbore.Rows.Count, allbore.ColumnNumber)
    allbore.Select ' выделить
```

```

For Each bore In allbore ' бежим по
    bore.Select ' выделяем ячейки
    If FindElement(bore.Value)
        borename.Add (bore)
    End If
Next bore
For Each elem In borename
    SelectBore (elem)
    Debug.Print (elem)
    For Each data In alldata
        Debug.Print (data)
    Next data
Next elem
End Sub

```

В окне отладки можно увидеть, что массив заполнен значениями соответствующими скважинам на данный момент.



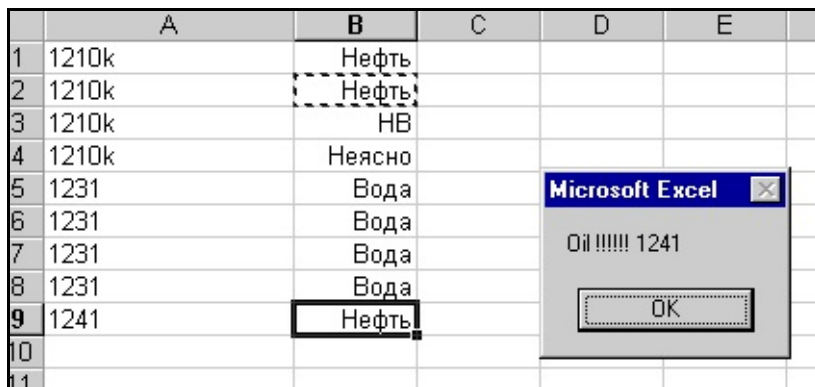
Ну, а теперь можно исследовать этот массив на пример решения какая скважина. Вообще то на самом деле я просто повторил работу БД. Сделал выборку. Давайте попробуем сделать простой вывод. Например если упоминается нефть но она нефтенасыщенная иначе пусто.

```
Sub FindOil()
```

.....

```
For Each elem In borename
    SelectBore (elem)
    Debug.Print (elem)
    For Each data In alldata
        'Debug.Print (data
        If data = "Нефть"
            MsgBox "Oi.
            Exit For
        End If
    Next data
Next elem
End Sub
```

При работе этой функции будет два окна для скважины с нефтью.



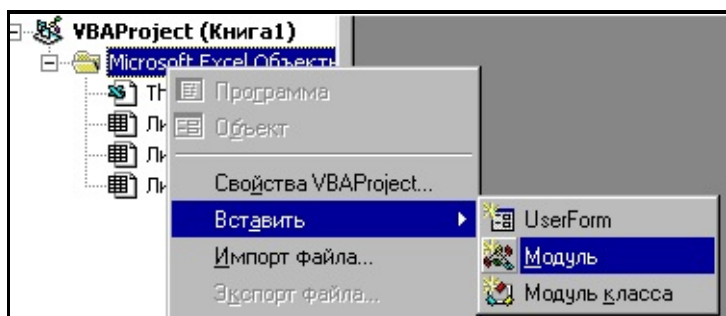
---

[Загрузить проект](#) | [Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 72 - Как создать свою функцию

Для того, чтобы создать и использовать свою функцию в **Excel** точно так же как стандартные функции Вам нужно перейти в редактор **VBA** читайте ["Шаг 1 - Первый макрос"](#). И добавить модуль.



Потом в модуле Вы создаете функцию:

```
Function Test(i As Integer) As Integer
    Test = i + 5
End Function
```

Все можно переходить на таблицу и использовать:

	C	D	E
= =Test(C3)			
	12	17	

Функция как родная.

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 73 - Выделенный диапазон Выше ячейки, второй метод

Добрый день!

Мне очень понравился Ваш сайт. Большое Вам за него спасибо! Особенно привлекательным его делает то, что различные среды программирования Вы рассматриваете не только по отдельности, но также разбираете связь между ними, например, написание **DLL** для Excel в **VC++**.

Однако, некоторые примеры вызывают возражения. Вот, скажем, задача выделить диапазон выше текущей ячейки (см. ["Шаг 65 - Выделение диапазона выше текущей ячейки"](#)). По-моему, слишком длинное решение такой простой задачи. Я бы это сделал так, воспользовавшись методом **Union**:

```
Sub SelectColumnData()  
    If ActiveCell.Row = 1 Then Exit Sub  
    If ActiveCell.Offset(-1, 0) <> "" Then  
        ActiveCell.Offset(-1, 0).Activate  
    If ActiveCell.Row = 1 Then Exit Sub  
    Do While Not ActiveCell.Offset(-1,  
        Union(Selection, ActiveCell)  
        If ActiveCell.Row = 1 Then  
    Loop  
End Sub
```

Задача очень простая, не правда ли, и решается четырьмя строками кода (за исключением трех **if**, которые отсекают выход за пределы листа). Переменные совсем не нужны, как это очень часто бывает в **VBA**.

Гуляев Александр Gulyaev@gw.tander.ru

---



[Предыдущий Шаг](#) Автор [Каев Артем](#).

## Шаг 74 - Инструкция Write #

Записывает неформатированные данные в файл с последовательным доступом.

**Write #номерФайла, [списокВывода]**

Параметры

**номерФайла**

Обязательный. Любой допустимый номер файла.

**списокВывода**

Необязательный. Одно или несколько разделяемых запятыми числовых выражений или строковых выражений, которые следует записать в файл.

Замечания

Данные, записанные с помощью инструкции **Write #**, обычно считываются из файла с помощью инструкции **Input #**.

Если аргумент списокВывода опущен, а после аргумента номерФайла идет только разделитель списка, в файл будет напечатана пустая строка. Для разделения выражений можно использовать пробелы, точки с запятой или запятые, которые в данной ситуации полностью эквивалентны.

Ниже приведены правила, которые используются при записи данных в файл с помощью инструкции **Write #**. Записанные данные могут быть корректно прочитаны с помощью инструкции **Input #** при наличии любой национальной настройки:

В качестве десятичного разделителя при записи числовых данных всегда используется точка.

При выводе логических данных (тип **Boolean**) в файл записываются ключевые слова **#TRUE#** или **#FALSE#**. Ключевые слова **True** и **False** не переводятся.

При выводе в файл данных типа **Date** используется универсальный формат даты. Если компонент, соответствующий дате или времени, отсутствует или равен нулю, в файл записывается только имеющийся в наличии компонент. Если аргумент списокВывода имеет значение **Empty**, в файл ничего не записывается.

Если списокВывода имеет значение **Null**, в файл записывается **#NULL#**.

Данные типа **Error** записываются в файл как **#ERROR кодОшибки#**. Ключевое слово **Error** не переводится.

В отличие от инструкции **Print #**, инструкция **Write #** вставляет запятые между элементами и заключает строки в кавычки по мере записи их в файл. Разработчику не требуется включать разделители в список явным образом. **Write #** вставляет символ новой строки, т.е. комбинацию символов возврата каретки и перевода строки (**Chr(13) + Chr(10)**), после записи в файл последнего символа, включенного в список Вывода.

Пример

В данном примере инструкция **Write #** используется для записи неформатированных данных в последовательный файл.

```
Open "TESTFILE" For Output As #1           ' (
Write #1, "Всем привет", 234              ' :
Write #1,                                  ' :

Dim MyBool, MyDate, MyNull, MyError
' Присваивает значения типа Boolean, Date,
MyBool = False : MyDate = #February 12, 19
MyError = CVErr(32767)
' Логические данные записываются как #TRUE:
' используется универсальный формат даты,
' соответствует 13 июля 1994 года. Значени
' #NULL#. Данные типа Error записываются к
Write #1, MyBool ; " - логическое значение
Write #1, MyDate ; " - дата"
Write #1, MyNull ; " - значение Null"
Write #1, MyError ; " - значение типа Erro
Close #1           ' Закрывает файл.
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 75 - Список операторов

[Оператор & Оператор \\*](#)

[Оператор +](#)

[Оператор -](#)

[Оператор /](#)

[Оператор \](#)

[Оператор ^](#)

[Оператор And](#)

[Операторы сравнения](#)

[Оператор Eqv](#)

[Оператор Imp](#)

[Оператор Is](#)

[Оператор Like](#)

[Оператор Mod](#)

[Оператор Not](#)

[Оператор Or](#)

[Оператор Xor](#)

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 76 - Старшинство операторов

Если выражение содержит несколько операторов, то значения компонентов выражения рассчитываются в определенном порядке. Такой порядок называют порядком старшинства или приоритетом операторов.

Если выражение содержит операторы разных типов, то первыми выполняются арифметические операции, следом за ними операции сравнения, а последними логические операции. Все операторы сравнения имеют равный приоритет, т.е. выполняются в порядке их расположения в выражении слева направо. Арифметические и логические операторы выполняются в порядке их расположения в следующей таблице:

Арифметические	Сравнения
Возведение в степень (^)	Равно
Изменение знака (-)	Не равняется
)	And
Умножение и деление (*, /)	Менее
Целое деление (\)	Больше
Деление по модулю (Mod)	Меньше или
Сложение и вычитание (+, -)	Больше
Слияние строк (&)	Li
	Is

Стоящие рядом в выражении операторы умножения и деления выполняются слева направо. В таком же порядке выполняются стоящие рядом операторы сложения и вычитания. Операторы внутри круглых скобок всегда выполняются раньше, чем операторы вне скобок. Порядок выполнения операторов, стоящих внутри скобок, определяется старшинством операторов.

Оператор конкатенации (слияния строк) (&) не является арифметическим оператором, однако, по порядку старшинства он следует сразу за арифметическими операторами и перед операторами сравнения.

Оператор **Like**, равный по старшинству остальным операторам сравнения, выделяется в самостоятельный тип оператора сравнения с образцом.

Оператор **Is** является оператором сравнения ссылок на объект. Этот оператор не выполняет сравнение объектов или их значений; он проверяет только, указывают ли две разные ссылки на один объект.

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 77 - Оператор &

Используется для слияния двух строковых выражений.

`результат = выражение1 & выражение2`

Параметры

**результат**

Обязательный; любая переменная типа **String** или **Variant**.

**выражение1**

Обязательный; любое выражение.

**выражение2**

Обязательный; любое выражение.

Замечания

Если выражение не содержит строковое значение, то оно преобразуется к подтипу **String** типа **Variant**. Если оба операнда являются строковыми выражениями, то результат имеет тип данных **String**; в противном случае результат принадлежит к подтипу **String** типа **Variant**. Если оба операнда является выражением со значением **Null**, результат имеет значение **Null**. Однако если значение **Null** имеет только одно выражение, то при слиянии с другим операндом этот операнд рассматривается как пустая строка (""). Любой операнд со значением **Empty** также рассматривается как пустая строка.

Пример

В данном примере демонстрируется использование оператора & для слияния строк.

```
Dim MyStr
MyStr = "ВсеМ" & " привет"
MyStr = "Check " & 123 & " Check"
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).



## Шаг 78 - Оператор \*

Возвращает произведение двух чисел.

результат = число1\*число2

Параметры

**результат**

Обязательный; любая числовая переменная.

**Число1**

Обязательный; любое числовое выражение.

**число2**

Обязательный; любое числовое выражение.

Замечания

результат обычно имеет тип данных выражения с максимальной точностью. В порядке возрастания точности следуют типы **Byte**, **Integer**, **Long**, **Single**, **Currency**, **Double** и **Decimal**. Из этого правила имеются такие исключения:

**Операция**

Умножение чисел типа **Single** и **Long**, результат должен получить значение подтипа **Long**, **Single** или **Date** типа **Variant**, выходя за рамки допустимых для этих типов диапазонов значений.

результат должен получить значение подтипа **Byte** типа **Variant**, выходящее за рамки допустимых для этого типа диапазона значений  
результат должен получить значение подтипа **Integer** типа **Variant**, выходящее за рамки допустимых для этого типа диапазона значений

Если один или оба операнда является выражением со значением **Null**, то

результат имеет значение **Null**. Выражение, имеющее значение **Empty**, рассматривается как имеющее значение 0.

Точность результатов операции умножения может отличаться от точности результатов операций сложения и вычитания.

Пример

В данном примере демонстрируется использование оператора \* для расчета произведения двух чисел.

```
Dim MyValue
MyValue = 2 * 2           ' Возвращает 2
MyValue = 459.35 * 334.90 ' Возвращает 153948.275
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 79 - Оператор +

Возвращает сумму двух чисел.

**результат = выражение1+выражение2**

Параметры

**результат**

Обязательный; любая числовая переменная.

**выражение1**

Обязательный; любое expression.

**выражение2**

Обязательный; любое выражение.

Замечания

Несмотря на то, что допускается использование оператора + для слияния строковых значений, рекомендуется во избежание возможной путаницы и необходимости дополнительных описаний в программе использовать для слияния строк оператор конкатенации &.

Если по крайней мере один из операндов не принадлежит к типу Variant, действуют следующие правила:

**Операнды**

Оба выражения имеют числовые типы данных (Integer, Long, Single, Double, Date, Currency).

Оба выражения имеют тип String

Одно выражение имеет числовой тип данных, а другое - Variant (за исключением значения Null).

Одно выражение имеет тип String, а другое - Variant (за исключением значения Null).

Одно выражение имеет тип Variant и содержит значение Null.

Одно выражение имеет числовой тип данных, а другое - тип String.

Оба выражения имеют значения Null

Если оба операнда являются выражениями типа **Variant**, действуют следующие правила:

### Операнды

Оба выражения типа **Variant** являются числовыми

Оба выражения типа **Variant** являются строковыми

Одно из выражений типа **Variant** является числовым

В простых операциях сложения с участием только операндов с числовыми типами результат обычно имеет тип данных слагаемого с максимальной точностью. В порядке возрастания точности следуют типы **Byte**, **Integer**, **Long**, **Single**, **Double**, **Currency** и **Decimal**. Из этого правила существуют такие исключения:

### Операция

Сложение чисел типа **Single** и **Long**.

результат должен получить значение подтипа **Date** типа **Variant**, выходящее за рамки , этих типов диапазонов значений.

результат должен получить значение подтипа **Variant**, выходящее за рамки допустимых для диапазона значений.

результат должен получить значение подтипа **Variant**, выходящее за рамки допустимых для диапазона значений.

Значение типа **Date** складывается со значени

Если один или оба операнда является выражением со значением **Null**, то результат имеет значение **Null**. Если оба операнда имеют значение **Empty**, результат имеет тип **Integer**. Если значение **Empty** имеет только один из операндов, результат получает значение второго операнда.

Точность результатов операций сложения и вычитания может отличаться от точности результатов операции умножения.

Пример

В данном примере демонстрируется использование оператора + для суммирования чисел. Оператор + позволяет также выполнять слияние строковых значений, однако, во избежание неопределенности рекомендуется выполнять эту операцию только с помощью оператора &. Если один из операндов содержит числовое значение, а второй строковое, то оператор + выполняет операцию сложения. Если оба операнда содержат строковые значения, выполняется слияние строк (конкатенация).

```
Dim MyNumber, Var1, Var2
MyNumber = 2 + 2           ' Возвращает 4
MyNumber = 4257.04 + 98112 ' Возвращает 98539.04

Var1 = "34": Var2 = 6     ' Инициализация Var1 = "34", Var2 = 6
MyNumber = Var1 + Var2    ' Возвращает 40

Var1 = "34": Var2 = "6"   ' Инициализация Var1 = "34", Var2 = "6"
MyNumber = Var1 + Var2    ' Возвращает "346"
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 80 - Оператор –

Возвращает разность двух чисел или изменяет знак числового выражения.

результат = число1–число2  
–число

Параметры

**результат**

Обязательный; любая числовая переменная.

**число**

Обязательный; любое числовое выражение.

**число1**

Обязательный; любое числовое выражение.

**число2**

Обязательный; любое числовое выражение.

Замечания

В первом варианте синтаксиса оператор – является арифметическим оператором вычитания, возвращающим разность двух чисел. Во втором варианте синтаксиса оператор – используется как унарный оператор, изменяющий знак выражения.

результат обычно имеет тип данных выражения с максимальной точностью. В порядке возрастания точности следуют типы **Byte**, **Integer**, **Long**, **Single**, **Double**, **Currency** и **Decimal**. Из этого правила имеются такие исключения:

**Операция**

Вычитание значений **Single** и **Long**.

результат должен получить значение подтипа **Single** или **Date** типа **Variant**, выходящее за допустимых для этих типов диапазонов значений  
результат должен получить значение подтипа типа **Variant**, выходящее за рамки допустимых значений этого типа диапазона значений.

результат должен получить значение подтипа типа Variant, выходящее за рамки допустимого этого типа диапазона значений.  
Значение типа Date участвует в операции вычисления со значением любого типа.  
Два значения типа Date участвуют в операции

Если один или оба операнда является выражением со значением **Null**, то результат имеет значение **Null**. Выражение со значением **Empty** рассматривается как имеющее значение 0.

Точность результатов операций сложения и вычитания может отличаться от точности результатов операции умножения.

Пример

В данном примере демонстрируется использование оператора - для вычисления разности двух чисел.

```
Dim MyResult
MyResult = 4 - 2           ' Возвращает 2
MyResult = 459.35 - 334.90 ' Возвращает 124.45
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 81 - Оператор /

Возвращает результат деления двух чисел.

результат = число1/число2

Параметры

**результат**

Обязательный; любая числовая переменная.

**число1**

Обязательный; любое числовое выражение.

**число2**

Обязательный; любое числовое выражение.

Замечания

результат обычно имеет тип данных **Double** или подтип **Double** типа **Variant**. Из этого правила имеются такие исключения:

Операция

Оба выражения имеют тип **Byte**, **Integer** или **Single**.

Оба выражения принадлежат к подтипу **Byte**, **Integer** или **Single** типа **Variant**.

В делении участвует значение типа **Decimal** и значение любого другого типа.

Если один или оба операнда является выражением со значением **Null**, то результат имеет значение **Null**. Выражение, имеющее значение **Empty**, рассматривается как имеющее значение 0.

Пример



В данном примере демонстрируется использование оператора / для деления двух чисел.

```
Dim MyValue  
MyValue = 10 / 4           ' Возвращает 2.5.  
MyValue = 10 / 3         ' Возвращает 3.333333333333333
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 82 - Оператор \

Возвращает результат целого деления двух чисел.

`результат = число1 \ число2`

Параметры

**результат**

Обязательный; любая числовая переменная.

**число1**

Обязательный; любое числовое выражение.

**число2**

Обязательный; любое числовое выражение.

Замечания

Перед выполнением деления значения операндов округляются до целых значений типа **Byte**, **Integer** или **Long**.

Обычно, результат имеет тип данных **Byte**, подтип **Byte** типа **Variant**, тип **Integer**, подтип **Integer** типа **Variant**, тип **Long** или подтип **Long** типа **Variant**, вне зависимости от того, является ли результат целым числом. Любая дробная часть отбрасывается. Если один или оба операнда является выражением со значением **Null**, результат имеет значение **Null**. Выражение, имеющее значение **Empty**, рассматривается как имеющее значение 0.

Пример

В данном примере демонстрируется использование оператора \ для целого деления двух чисел.

```
Dim MyValue
MyValue = 11 \ 4           ' Возвращает 2.
MyValue = 9 \ 3           ' Возвращает 3.
MyValue = 100 \ 3        ' Возвращает 33.
```

Автор [Каев Артем](#).

## Шаг 83 - Оператор ^

Возводит число в степень.

**результат = число^показатель**

Параметры

**результат**

Обязательный; любая числовая переменная.

**число**

Обязательный; любое числовое выражение.

**показатель**

Обязательный; любое числовое выражение.

Замечания

Отрицательные значения операнда число допускаются только для целых значений операнда показатель. Стоящие рядом в выражении несколько операторов возведения в степень ^ выполняются слева направо.

Обычно, результат имеет тип данных **Double** или **Variant**, содержащий значение типа **Double**. Если любой из операндов число или показатель представлен выражением, имеющим значение **Null**, то результат также имеет значение **Null**.

Пример

В данном примере демонстрируется использование оператора ^ для возведения числа в степень.

```
Dim MyValue
MyValue = 2 ^ 2           ' Возвращает 4.
MyValue = 3 ^ 3 ^ 3      ' Возвращает 19683
MyValue = (-5) ^ 3       ' Возвраща
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 84 - Оператор And

Возвращает результат конъюнкции (логического И) для двух выражений.

`результат = выражение1 And выражение2`

Параметры

**результат**

Обязательный; любая числовая переменная.

**выражение1**

Обязательный; любое выражение.

**выражение2**

Обязательный; любое выражение.

Замечания

Если оба выражения имеют значение **True**, результат имеет значение **True**. Если хотя бы одно из выражений имеет значение **False**, результат имеет значение **False**. Возвращаемые оператором значения представлены в следующей таблице:

выражение1	выражение2	результат
True	True	True
True	False	False
True	Null	Null
False	True	False
False	False	False
False	Null	False
Null	True	Null
Null	False	False
Null	Null	Null

Оператор **And** выполняет также поразрядное сравнение двух числовых выражений и присваивает соответствующему разряду в переменной результат следующее значение:

Разряд в выражение1	Разряд в выражение:
0	0
0	1
1	0
1	1

### Пример

В данном примере демонстрируется использование оператора **And** для выполнения операции конъюнкции (логического умножения) двух выражений.

```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null
MyCheck = A > B And B > C
MyCheck = B > A And B > C
MyCheck = A > B And B > D
MyCheck = A And B
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 85 - Операторы сравнения

Используются для сравнения двух выражений.

```
результат = выражение1 операторСравнения в  
результат = объект1 Is объект2  
результат = строка Like образец
```

Параметры

**результат**

Обязательный; любая числовая переменная.

**выражение**

Обязательный; любое выражение.

**операторСравнения**

Обязательный; любой оператор сравнения.

**объект**

Обязательный; любое имя объекта.

**строка**

Обязательный; любое строковое выражение.

**образец**

Обязательный; любое строковое выражение или диапазон символов.

Замечания

Ниже приведен список операторов сравнения и условия, при которых результат имеет значение **True**, **False** или **Null**:

Оператор	True
< (Меньше)	выражение1 < выражение2
<= (Меньше или равняется)	выражение1 <= выражение2
> (Больше)	выражение1 > выражение2
>= (Больше или равняется)	выражение1 >= выражение2
= (Равняется)	выражение1 = выражение2
<> (Не равняется)	выражение1 <> выражение2

Операторы **Is** и **Like** функционально отличаются от операторов сравнения, перечисленных в таблице.

При сравнении двух выражений иногда затруднительно определить, будут ли сравниваться выражения как числовые или как строковые. В следующей таблице показано, как сравниваются выражения, если хотя бы одно из них не принадлежит к типу **Variant**:

#### Операнды

Оба выражения имеют числовые типы данных (Integer, Long, Single, Double, Date, Currency или Decimal).

Оба выражения имеют тип String.

Одно выражение имеет числовой тип, а другое — строковый, содержащий число или допускающий преобразование в число.

Одно выражение имеет числовой тип, а другое — строковый, не допускающий преобразования в число.

Одно выражение имеет тип String, а другое — Variant.

Одно выражение имеет значение Empty, а другое — Variant.

Одно выражение имеет значение Empty, а другое — строковое.

Если оба операнда, выражение1 и выражение2, принадлежат к типу **Variant**, то способ сравнения определяется подтипами, к которым они принадлежат. В следующей таблице показано, как осуществляется сравнение в зависимости от подтипов типа **Variant**:

#### Операнды

Оба выражения типа Variant имеют числовые значения.

Оба выражения типа Variant имеют строковые значения.

Одно из выражений типа Variant является числовым, а другое — строковым.

Одно из выражений типа Variant имеет значение Empty, а другое — числовое значение.

Одно из выражений типа Variant имеет значение Empty, а другое — строковое значение.

Оба выражения типа Variant имеют значение Empty.



При сравнении значения типа **Single** со значением типа **Double**, значение типа **Double** округляется до точности типа **Single**.

Если значение типа **Currency** сравнивается со значением типа **Single** или **Double**, то значение типа **Single** или **Double** преобразуется к типу **Currency**. Аналогично, при сравнении значения типа **Decimal** со значением типа **Single** или **Double**, значение типа **Single** или **Double** преобразуется к типу **Decimal**. Для типа **Currency** любая дробная часть со значением, меньшим 0,0001 может оказаться потерянной. Для типа **Decimal** могут быть потеряны дробные части, меньшие **1E-28**, или возникнуть ошибка переполнения. Потеря дробных частей может привести к тому, что числа, не являющиеся целыми, будут сравниваться как целые.

Пример

В данном примере демонстрируется использование операторов сравнения для сравнения выражений.

```
Dim MyResult, Var1, Var2
MyResult = (45 < 35)           ' Возвращает True
MyResult = (45 = 45)          ' Возвращает True
MyResult = (4 <> 3)           ' Возвращает True
MyResult = ("5" > "4")       ' Возвращает True

Var1 = "5": Var2 = 4          ' Инициализация
MyResult = (Var1 > Var2)     ' Возвращает True

Var1 = 5: Var2 = Empty
MyResult = (Var1 > Var2)     ' Возвращает True

Var1 = 0: Var2 = Empty
MyResult = (Var1 = Var2)     ' Возвращает True
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 86 - Оператор Eqv

Используется для проверки логической эквивалентности двух выражений.

результат = выражение1 Eqv выражение2

Параметры

**результат**

Обязательный; любая числовая переменная.

**выражение1**

Обязательный; любое выражение.

**выражение2**

Обязательный; любое выражение.

Замечания

Если хотя бы одно из выражений имеет значение **Null**, то результат также имеет значение **Null**. Если ни одно из выражений не имеет значение **Null**, результат определяется следующим образом:

выражение1	выражение2	результат
True	True	True
True	False	False
False	True	False
False	False	True

Оператор **Eqv** выполняет также поразрядное сравнение двух числовых выражений и присваивает соответствующему разряду в переменной результат следующее значение:

Разряд в выражение1	Разряд в выражение2
0	0
0	1
1	0
1	1

## Пример

В данном примере демонстрируется использование оператора **Eqv** для выполнения операции логической эквивалентности двух выражений.

```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null ' Инициализация
MyCheck = A > B Eqv B > C      ' |
MyCheck = B > A Eqv B > C      ' |
MyCheck = A > B Eqv B > D      ' |
MyCheck = A Eqv B               ' |
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 87 - Оператор Imp

Выполняет операцию логической импликации для двух выражений.

результат = выражение1 Imp выражение2

Параметры

**результат**

Обязательный; любая числовая переменная.

**выражение1**

Обязательный; любое выражение.

**выражение2**

Обязательный; любое выражение.

Замечания

Возвращаемые оператором значения представлены в следующей таблице:

выражение1	выражение2	результат
True	True	True
True	False	False
True	Null	Null
False	True	True
False	False	True
False	Null	True
Null	True	True
Null	False	Null
Null	Null	Null

Оператор **Imp** выполняет также поразрядное сравнение двух числовых выражений и присваивает соответствующему разряду в переменной результат следующее значение:

Разряд в выражение1  
0

Разряд в выражение:  
0

0	1
1	0
1	1

### Пример

В данном примере демонстрируется использование оператора Imp для выполнения операции логической импликации двух выражений.

```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null ' Инициализация
MyCheck = A > B Imp B > C
MyCheck = A > B Imp C > B
MyCheck = B > A Imp C > B
MyCheck = B > A Imp C > D
MyCheck = C > D Imp B > A
MyCheck = B Imp A
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 88 - Оператор Is

Используется для сравнения двух переменных, содержащих ссылки на объекты.

**результат = объект1 Is объект2**

Параметры

**результат**

Обязательный; любая числовая переменная.

**объект1**

Обязательный; любое имя объекта.

**объект2**

Обязательный; любое имя объекта.

Замечания

Если переменные объект1 и объект2 содержат ссылки на один объект, результат имеет значение **True**; в противном случае результат имеет значение **False**. Ссылки на один объект в двух переменных задаются различными способами.

В следующей инструкции в переменной A задается ссылка на тот же объект, что и в переменной B:

**Set A = B**

Следующая инструкция задает в переменных A и B ссылку на тот же объект, что и в переменной C:

**Set A = C**

**Set B = C**

Пример

В данном примере демонстрируется использование оператора **Is** для сравнения ссылок на объекты. Имена объектных переменных являются произвольными и используются только для демонстрации.

```
Dim MyObject, YourObject, ThisObject, OtherObject
Set YourObject = MyObject
Set ThisObject = MyObject
Set ThatObject = OtherObject
MyCheck = YourObject Is ThisObject
MyCheck = ThatObject Is ThisObject
MyCheck = MyObject Is ThatObject
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 89 - Оператор Like

Используется для сравнения двух строковых выражений.

результат = строка Like образец

Параметры

**результат**

Обязательный; любая числовая переменная.

**строка**

Обязательный; любое строковое выражение.

**образец**

Обязательный; любое строковое выражение, удовлетворяющее описанным ниже соглашениям о сравнениях строк с образцом.

Замечания

Если значение переменной строка соответствует значению переменной образец, результат имеет значение **True**; при их несоответствии результат имеет значение **False**. Если любой из операндов, строка или образец, имеет значение **Null**, результат также имеет значение **Null**.

Характеристики оператора **Like** определяются настройкой, задаваемой в инструкции **Option Compare**. Если не указано другое, то стандартный способ сравнения строк в каждом модуле задается инструкцией **Option Compare Binary**.

Инструкция **Option Compare Binary** указывает способ сравнения строковых значений, основанный на порядке сортировки, который определяется внутренним двоичным представлением символов. В **Microsoft Windows** порядок сортировки определяется кодовой страницей. Ниже демонстрируется типичный пример упорядочивания символов:

A < B < E < Z < a < b < e < z < A < K < Ш .

Инструкция **Option Compare Text** указывает способ сравнения строковых значений, основанный на порядке сортировки по алфавиту без учета регистра, который определяется национальной настройкой системы.



Приведенная выше последовательность символов при сортировке с настройкой **Option Compare Text** будет упорядочиваться следующим образом:

(A=a) < (B=b) < (E=e) < (Z=z) < (A=a) < (K:

Существуют встроенные средства сравнения строковых значений, допускающие использование в образце подстановочных знаков, списков символов и диапазонов символов. В следующей таблице перечислены специальные символы, используемые в переменной образец, и соответствующие им символы в переменной строка:

Специальные символы	Совпадающие символы
?	Любой одиночный си
*	Любое количество с
#	Любая одиночная ци
[список]	Любой один
[!список]	Любой один

Любой одиночный символ в переменной строка может быть сравнен с группой из одного или нескольких символов (список), заключенных в квадратные скобки ([ ]). Такая группа может содержать символы практически с любыми кодами, включая цифры.

Специальные символы левая квадратная скобка ([), вопросительный знак (?), символ (#) и звездочка (\*) сравниваются как обычные символы, только если они заключены в квадратные скобки. Правая квадратная скобка (]) в группе всегда рассматривается как специальный символ, но может сравниваться как обычный символ вне группы.

В дополнение к простому списку символов в квадратных скобках, допускается указание в образце диапазона символов с помощью дефиса (-), отделяющего нижнюю и верхнюю границы диапазона. Например, [A-Z] указывает, что совпадающими будут считаться все символы верхнего регистра латинского алфавита, находящиеся в нужной позиции в переменной строка. Допускается указание внутри прямых скобок нескольких диапазонов без символов разделителей.

Конкретный вид указанного диапазона определяется действующим во

время выполнения способом упорядочивания символов (который задается инструкцией **Option Compare** и национальной настройкой системы). Если использовать приведенный выше пример в режиме **Option Compare Binary**, то диапазону [A–E] соответствуют символы A, B и E. В режиме **Option Compare Text**, образцу [A–E] соответствуют символы **A, a, B, b, E, e**. Отметим, что в это диапазон не попадают символы K или k, поскольку все символы русского алфавита размещаются в порядке сортировки после стандартных символов латинского алфавита.

Существуют также следующие правила, действующие при сравнении с образцом:

Восклицательный знак (!) в начале перечня список указывает, что совпадающим с образцом считается любой символ в соответствующей позиции в операнде строка, за исключением входящих в список. Вне квадратных скобок восклицательный знак рассматривается как обычный символ.

Дефис (–) в начале списка или сразу после восклицательного знака, а также в конце списка рассматривается как обычный символ. В любой другой позиции внутри квадратных скобок он рассматривается как определение диапазона.

Определять диапазон необходимо в порядке возрастания. [A–Z] является допустимым определением диапазона, а [Z–A] недопустимым.

Пустая пара квадратных скобок [] игнорируется; она рассматривается как пустая строка ("").

В некоторых языках имеются специальные символы алфавита, представляющие пару букв. Например, символ "ж" представляет стоящие рядом буквы "а" и "е". Оператор Like распознает такой символ и соответствующую пару букв как эквивалентные.

Если в системных национальной настройке указан язык, в котором используются специальные символы, то включение такого символа и в переменную образец, и в переменную строка приводит к тому, что эквивалентная пара в букв в другом операнде рассматривается как совпадающая с этим символом. Аналогично, при включении специального символа в образец внутри квадратных скобок (одного символа или как часть определения списка или диапазона) он представляет эквивалентную пару букв в операнде строка.

Пример

В данном примере демонстрируется использование оператора **Like** для сравнения строкового значения с образцом.

```
Dim MyCheck
MyCheck = "aBBBa" Like "a*a"           ' |
MyCheck = "F" Like "[A-Z]"
MyCheck = "F" Like "[!A-Z]"
MyCheck = "a2a" Like "a#a"
MyCheck = "aM5b" Like "a[L-P]#[!c-e]"  ' |
MyCheck = "BAT123khg" Like "B?T*"
MyCheck = "CAT123khg" Like "B?T*"
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 90 - Оператор Mod

Возвращает остаток при целом делении двух чисел (значение по модулю).

`результат = число1 Mod число2`

Параметры

**результат**

Обязательный; любая числовая переменная.

**число1**

Обязательный; любое числовое выражение.

**число2**

Обязательный; любое числовое выражение.

Замечания

Выполняется деление по модулю (нахождение остатка) число1 на число2 (с округлением действительных чисел до целых). результат получает значение остатка. Например, в следующем выражении A (результат) равняется 5.

`A = 19 Mod 6.7`

Обычно, результат имеет тип данных **Byte**, подтип **Byte** типа **Variant**, тип **Integer**, подтип **Integer** типа **Variant**, тип **Long** или подтип **Long** типа **Variant**, вне зависимости от того, является ли результат целым числом. Любая дробная часть отбрасывается. Если один или оба операнда является выражением со значением **Null**, результат имеет значение **Null**. Выражение, имеющее значение **Empty**, рассматривается как имеющее значение 0.

Пример

В данном примере оператор **Mod** выполняет деление двух чисел и возвращает остаток. Если любой из операндов является действительным числом, он предварительно округляется до целого.

```
Dim MyResult
MyResult = 10 Mod 5      ' Возвращает 0.
```

```
MyResult = 10 Mod 3      ' Возвращает 1.  
MyResult = 12 Mod 4.3   ' Возвращает 0.  
MyResult = 12.6 Mod 5   ' Возвращает 3.
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 91 - Оператор Not

Выполняет над выражением операцию логического отрицания.

`результат = Not выражение`

Параметры

**результат**

Обязательный; любая числовая переменная.

**выражение**

Обязательный; любое выражение.

Замечания

Возвращаемые оператором значения представлены в следующей таблице:

выражение	результат
True	False
False	True
Null	Null

Оператор **Not** выполняет также поразрядное изменение значений каждого разряда переменной и присваивает соответствующему разряду в переменной результат следующее значение:

Разряд в выражение	результат
0	1
1	0

Пример

В данном примере демонстрируется использование оператора **Not** для выполнения операции логического отрицания для одного выражения.

```
Dim A, B, C, D, MyCheck  
A = 10: B = 8: C = 6: D = Null
```

MyCheck = Not(A > B)

MyCheck = Not(B > A)

MyCheck = Not(C > D)

MyCheck = Not A



---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 92 - Оператор Or

Выполняет операцию логического ИЛИ (сложения) для двух выражений.

`результат = выражение1 Or выражение2`

Параметры

**результат**

Обязательный; любая числовая переменная.

**выражение1**

Обязательный; любое выражение.

**выражение2**

Обязательный; любое выражение.

Замечания

Если одно или оба выражения истинны (имеют значение **True**), результат также имеет значение **True**. Возвращаемые оператором значения представлены в следующей таблице:

выражение1	выражение2	результат
True	True	True
True	False	True
True	Null	True
False	True	True
False	False	False
False	Null	Null
Null	True	True
Null	False	Null
Null	Null	Null

Оператор **Or** выполняет также поразрядное сравнение двух числовых выражений и присваивает соответствующему разряду в переменной результат следующее значение:



Разряд в выражение1	Разряд в выражение:
0	0
0	1
1	0
1	1

### Пример

В данном примере демонстрируется использование оператора **Or** для выполнения операции дизъюнкции (логического сложения) двух выражений.

```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null ' Инициализация
MyCheck = A > B Or B > C      ' True
MyCheck = B > A Or B > C      ' False
MyCheck = A > B Or B > D      ' True
MyCheck = B > D Or B > A      ' False
MyCheck = A Or B              ' True
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 93 - Оператор Xor

Выполняет операцию исключающего ИЛИ для двух выражений.

[результат =] выражение1 Xor выражение2

Параметры

**результат**

Обязательный; любая числовая переменная.

**выражение1**

Обязательный; любое выражение.

**выражение2**

Обязательный; любое выражение.

Замечания

Если одно и только одно из выражений истинно (имеет значение **True**), результат имеет значение **True**. Если хотя бы одно из выражений имеет значение **Null**, результат также имеет значение **Null**, Если ни одно из выражений не имеет значение **Null**, результат определяется следующим образом:

выражение1	выражение2	результат
True	True	False
True	False	True
False	True	True
False	False	False

Оператор Xor выполняет также поразрядное сравнение двух выражений и присваивает соответствующему разряду в переменной результат следующее значение:

Разряд в выражение1	Разряд в выражение2
0	0
0	1



## Шаг 73 - Инструкция With

Выполняет последовательность инструкций над одиночным объектом или определяемым пользователем типом данных.

```
With объект  
[инструкции]  
End With
```

Параметры

**объект**

Обязательный. Имя объекта или определяемого пользователем типа.

**инструкции**

Необязательный. Одна или несколько инструкций, выполняемых над объектом.

Замечания

Инструкция **With** позволяет выполнить последовательность инструкций над указанным объектом, не повторяя задание имени объекта. Например, если имеется несколько свойств, которые необходимо изменить для одиночного объекта, то удобнее поместить инструкции присвоения свойств внутрь управляющей структуры **With**, указав ссылку на объект один раз, вместо того, чтобы ссылаться на объект при каждом присвоении его свойств. Следующий пример демонстрирует использование инструкции **With** для присвоения значений нескольким свойствам одного объекта.

```
With MyLabel  
    .Height = 2000  
    .Width = 2000  
    .Caption = "Объект MyLabel"  
End With
```

После входа в блок **With** значение аргумента объект изменить невозможно. В результате, невозможно использование одной инструкции **With** для воздействия на различные объекты.

Допускается создание вложенных конструкций **With** внутри другого. Однако, поскольку ссылки на объекты внешних блоков **With** перекрываются во внутренних блоках, необходимо во внутренних блоках **With** задавать полные ссылки на любой компонент объекта из внешнего блока **With**.

Нельзя выполнять переходы внутрь или из блоков **With**. Если не выполнены инструкции **With** или **End With**, возможно возникновение ошибок или непредсказуемое поведение объектов.

Пример

В данном примере инструкция **With** используется для выполнения набора инструкций над одним объектом. Объект **MyObject** и его свойства используются исключительно в иллюстративных целях.

```
With MyObject
    .Height = 100
    .Caption = "Привет"
    With .Font
        .Color = Red
        .Bold = True
    End With
End With
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 72 - Инструкция Width #

Задаёт ширину строки для файла, открытого с помощью инструкции **Open**.

`Width #номерФайла, ширина`

Параметры

**номерФайла**

Обязательный. Любой допустимый номер файла.

**ширина**

Обязательный. Числовое выражение в интервале от 0–255 включительно, которое указывает, сколько символов следует поместить на одной строке прежде, чем перейти к следующей строке. Если ширина равняется 0 (это значение используется по умолчанию), ширина строки не ограничивается. Аргумента ширина по умолчанию имеет значение 0.

Пример

```
Dim I
Open "TESTFILE" For Output As #1
Width #1, 5
For I = 0 To 9
    Print #1, Chr(48 + I);
Next I
Close #1
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).

## Шаг 71 - Инструкция While...Wend

Выполняет последовательность инструкций, пока заданное условие имеет значение **True**.

```
While условие  
[инструкции]  
Wend
```

Параметры

**условие**

Обязательный. Числовое выражение или строковое выражение, которое имеет значение **True** или **False**. Если условие имеет значение **Null**, условие рассматривается как имеющее значение **False**.

**инструкции**

Необязательный. Одна или несколько инструкций, выполняемых, пока условие имеет значение **True**.

Замечания

Если условие имеет значение **True**, выполняются все инструкции до инструкции **Wend**. Затем управление возвращается инструкции **While** и вновь проверяется условие. Если условие по-прежнему имеет значение **True**, процесс повторяется. Если оно не имеет значение **True**, выполнение возобновляется с инструкции, следующей за инструкцией **Wend**.

Циклы **While...Wend** могут иметь любую глубину вложенности. Каждая инструкция **Wend** соответствует предшествующей инструкции **While**.

Инструкция **Do...Loop** обеспечивает более структурированный и гибкий способ организации циклов.

Пример

В данном примере инструкция **While...Wend** используется для увеличения переменной-счетчика. Инструкции в цикле выполняются до тех пор, пока указанное условие не **True**.

```
Dim Counter
```

```
Counter = 0           ' Инициализация
While Counter < 20   ' Цикл
    Counter = Counter + 1 ' Увеличиваем
Wend                 ' Завершаем
                    ' если Counter
Debug.Print Counter  ' Выводит
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).



## Шаг 70 - Инструкция Unload

Удаляет объект из памяти.

### Unload объект

Параметры

#### объект

Обязательный прототип объект представляет собой объектное выражение, значением которого является один из объектов, перечисленных в списке "Применение".

Замечания

Когда объект выгружается, он удаляется из памяти, и вся память связанная с объектом, распределяется заново. Пока объект опять не помещен в память с помощью инструкции **Load**, пользователь не может взаимодействовать с ним, и взаимодействие ни программным образом также невозможно.

Пример

В следующем примере предполагается, что в программе определены пользователем две формы **UserForms**. Если произошло событие "Инициализация" (**Initialize**) для формы **UserForm1**, форма **UserForm2** загружается и выводится на экран. Когда пользователь выбирает при помощи мыши **UserForm2**, она выгружается, и появляется форма **UserForm1**. Если же выбирается **UserForm1**, то она, в свою очередь, выгружается.

```
' Процедура события "Инициализация" (Initi:  
' для формы UserForm1.  
Private Sub UserForm_Initialize()  
    Load UserForm2  
    UserForm2.Show  
End Sub  
  
' Событие "Нажатие кнопки" (Click) для фор
```

```
Private Sub UserForm_Click()  
    Unload UserForm2  
End Sub
```

```
' Событие "Нажатие кнопки" (Click) для фор  
Private Sub UserForm_Click()  
    Unload UserForm1  
End Sub
```

---

[Предыдущий Шаг](#) | [Следующий Шаг](#)

Автор [Каев Артем](#).