Научно-внедренческий центр «ГеоС»

Комплекс программ для проектирования мебели

КЗ

Версия 6.0

Руководство пользователя по написанию сценариев

Нижний Новгород
2008

Содержание

Введение

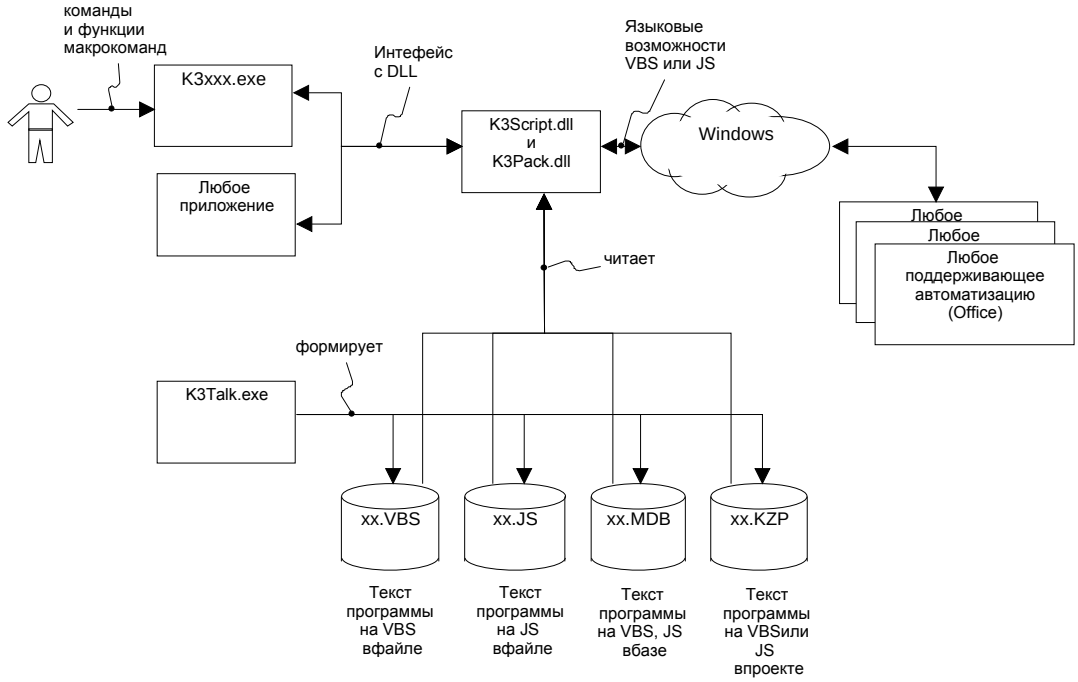
В текущей версии **К3** добавлена возможность использовать программы - сценарии (скрипты - scripts), написанные на языке **VBScript** (**VBS**) или **Jscript** (**JS**).

Основное назначение использования сценариев - это дать возможность системе **К3** взаимодействовать с другими приложениями, поддерживающими автоматизацию.

Данное руководство не предназначено для детального описания языков. Описание и способы работы с ними описаны в соответствующих руководствах. Один из подробных справочников находится в справочной системе **Script56.chm**. Данный файл поставляется в комплекте с системой и находится на дистрибутивном диске.

Для успешной работы со сценариями необходимо иметь в папке **BIN** динамические библиотеки **K3Script30.dll**, **K3Pack.dll.**

Библиотека **K3Script30.dll** является связующим звеном системы **К3** с внешним миром. Схематически это можно представить себе следующим образом:



Автоматизация приложений

На рисунке 4 представлены все основные компоненты, участвующие в процессе работы со сценариями с точки зрения пользователя системы **К3**.

В общем случае весь процесс можно описать простым набором операций:

- Создание сценария;
- Отладка сценария;
- Сохранение сценария в специальном виде (необязательно);
- Выполнение сценария из системы **К3.**

Принципиально этот механизм можно внедрить в любое приложение. Для этого достаточно иметь две динамические библиотеки и описание экспортируемых функций.

Сценарий - это, фактически, текст программы, написанной на одной из разновидностей **VBS** или **JS**.

В качестве редактора текстов сценариев можно выбрать:

- Любой доступный редактор
- Специализированный редактор **К3Talk** из системы **К3**

Как правило, любой текст храниться в каком либо файле с определенным расширением. Для сценариев принято расширение **VBS** или **JS**. Таким образом, каждой программе (сценарию) будет соответствовать свой файл. Группу функций в пределах одного файла в дальнейшем будем называть модулем.

Для уменьшения количества файлов и более удобной систематизации, в системе **K3** предлагается хранить сценарии в специализированной базе данных или в проектах. В этом случае необходимо использовать специализированный редактор **K3Talk**.

В качестве общего обзора приведём характеристики каждого из способов хранения:

Тип	Хранение множества модулей	Запоминание параметров вызова	Возможность вызова функций из других модулей	Хранение разноязыковых модулей	Скрытие текста программы
Хранение в файле VBS или JS	Нет	Нет	Нет	Нет	Нет
Хранение в базе MDB	Да	Да	Ограничено	Да	Нет
Хранение в проекте	Да	Да	Да	Нет	Да

Способы хранения сценариев

Сценарии

В данной главе изложены основные принципы и терминология, принятые при написании сценариев в **КЗ.**

Использование сценариев

Предположим, что в нашем распоряжении имеются уже созданные кем-то тексты сценариев. Пусть, для начала сценарий находится в файле **Test.vbs**.

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

Function Celcius12(fDegrees, fCr)
  Celcius12 = (fDegrees - 32) * 5 / 9
  Celcius12 = Celcius12*fCr
End Function

Function ShowExcel(obj)
  obj.Visible = True
End Function

Sub Test()
Dim obj
  Set obj = CreateObject("Excel.Application")
ShowExcel(obj)
  obj.Quit
  Set obj = Nothing
End Sub
```

Первое, что можно заметить – это то, что сценарий состоит из набора функций и подпрограмм. И второе – то, что нет явно выделенной главной функции. Так же видно, что функции могут принимать параметры и возвращать результаты работы. Функции в пределах сценария могут вызывать друг друга. Такой текст в дальнейшем будем называть модулем.

Таким образом, для того, что бы использовать сценарий, нам необходимо использовать:

- Имя файла со сценарием;
- Имя первой выполняемой функции;
- Параметры, передаваемые в функцию (если таковые необходимы).

В **К3** есть команда `Script`, позволяющая исполнять сценарии. Формат команды:

Script VBS, *<FileName>*, *<FuncName>*, [*<Params>*]

где VBS – модификатор, указывающий, что в текущий момент времени работать будем со сценарием, находящимся в файле **VBS,** *<FileName>* - полное имя файла, *<FuncName>* – имя функции, являющейся точкой входа, *<Params>* – текстовая строка с набором переменных, разделенных запятой и передаваемых в виде параметров.

Для исполнения сценария достаточно выполнить из командной строки (или макрокоманды) следующую команду:

```
script VBS, "d:\\cel.vbs","Celcius","12"
```

То есть, выполнить сценарий, расположенный в отдельном файле, имя этого файла **d:\\cel.vbs**. Найти в нем функцию **Celsius**, передать ей в качестве параметра число **12** и выполнить.

Команды **К3** не возвращают значений, но обладают одним свойством - Подсказывают каждый следующий шаг.

Например, можно ввести:

```
script
```

Система запросит тип:

```
Тип:
VBS
```

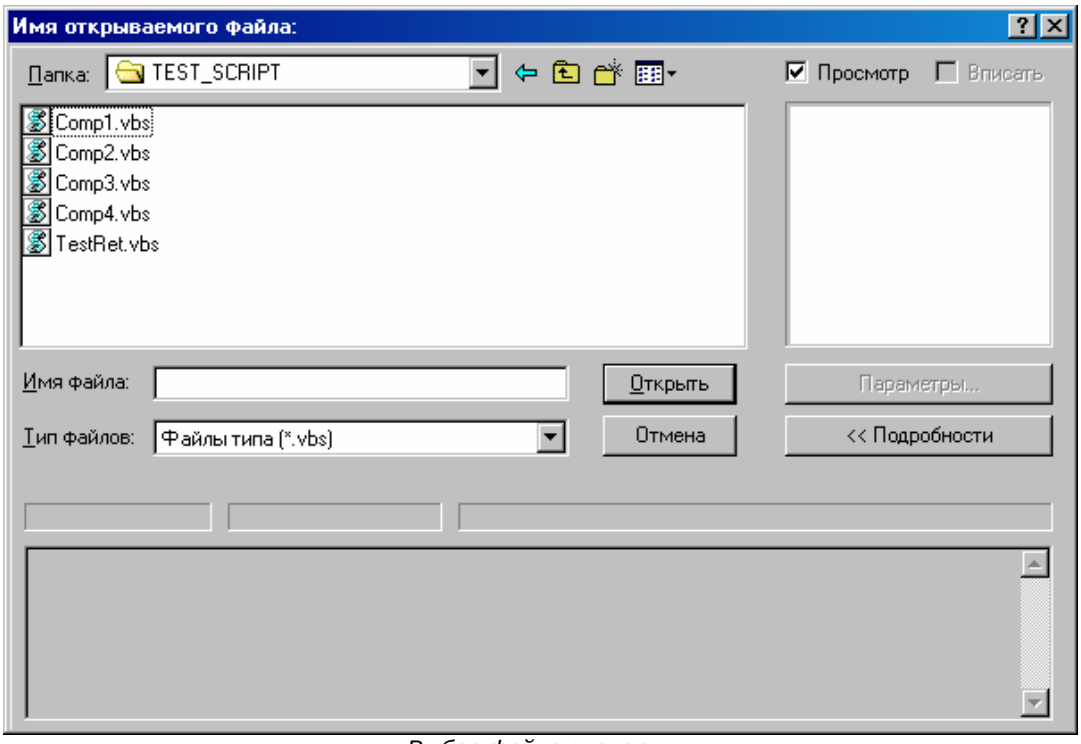
Далее надо определиться с файлом

```
Файл VBS:
```

Здесь либо можно указать полный путь до требуемого файла в кавычках:

```
Файл VBS:
"D:\\Test.vbs"
```

Либо просто нажать **<ENTER>**, тогда представиться возможность выбрать файл из диалогового окна [9]:



Выбор файла сценария

Далее пойдёт запрос на имя функции и требуемых параметров. Аналогичным образом работают все нижеприведённые команды (но не функции). В зависимости от типа языка и его способа хранения подсказки могут изменяться.

Для того, что бы получить возвращаемое значение, необходимо воспользоваться функцией. Формат функции:

VARIANT **fvbsscript** (STRING <*FileName*>, STRING <*FuncName*>[,STRING <*Params*>])

где <*FileName*> - полное имя файла, <*FuncName*> – имя функции, являющейся точкой входа, <*Params*> – текстовая строка с набором переменных, разделенных запятой и передаваемых в виде параметров.

При использовании того же сценария, но с помощью вызова функции, система вернет значение. Например:

```
=fvbsscript("d:\\cel.vbs","Celsius","12")
-11.111
```

Для того, чтобы воспользоваться сценарием, написанным на **JavaScript** используется та же команда, но с другим признаком:

Script JS, *<FileName>*, *<FuncName>*, [*<Params>*]

где JS – модификатор, указывающий, что в текущий момент времени работать будем со сценарием, находящимся в файле **JS,** *<FileName>* - полное имя файла, *<FuncName>* – имя функции, являющейся точкой входа, *<Params>* – текстовая строка с набором переменных, разделенных запятой и передаваемых в виде параметров.

Для исполнения сценария достаточно выполнить из командной строки (или макрокоманды) следующую команду:

```
script JS, "d:\\cel.js","Celcius","12"
```

То есть, выполнить сценарий расположенный в отдельном файле, имя этого файла **d:\\cel.js**. Найти в нем функцию **Celsius**, передать ей в качестве параметра число **12** и выполнить.

Для того, что бы получить возвращаемое значение, необходимо воспользоваться функцией. Формат функции:

VARIANT **FJsScript** (STRING *<FileJs>*, STRING *<FuncName>*, STRING *<Params>*)

Функция инициирует исполнение функции с именем *<FuncName>* в сценарии с полным именем файла *<FileJs>* написанном в отдельном модуле на языке **JavaScript**. Параметр *<Params>* – строка со значениями параметров, передаваемыми функции, разделенными запятой.

Пример:

```
=FJsScript("d:\\cel.js","Celsius","12");
 -11.111
=FJsScript("d:\\lec.js","Func1","12,15,23")
 152
```

Подробнее синтаксис команд и функций работы со сценариями описан в **Инструкции по макропрограммированию.**

Использование сценариев из набора (базы данных)

Вполне вероятно, что количество файлов **VBS** и **JS** будет неуклонно расти. Для того чтобы минимизировать количество файлов в системе и ввести для них систематизацию, предлагается тексты сценариев хранить в специальной базе - базе сценариев.

В качестве хранилища выбрана база формата **Access**. Это файлы с расширением **MDB**.

Предположим, что у нас уже существует такая база. Естественно, вводя промежуточный объект, формат команд претерпевает изменения.

Команда, выполняющая сценарий из базы данных, будет иметь следующий формат:

Script MDB *<FileMdb>*, *<ScriptName>*, *<FuncName>*, [*<Params>*]

Команда вызывает на выполнение функцию с именем *<FuncName>* в сценарии, находящемся в модуле *<ScriptName>* в сборке сценариев в файле с именем *<FileMdb>*. *<Params>* – текстовая строка с набором переменных, разделенных запятой, передаваемых в виде параметров. Как видно, появился дополнительный параметр – имя сценария. Это связано с тем, что внутри одной базы данных может находиться множество сценариев. Именно поэтому множество именованных сценариев, находящихся в одной базе данных, в дальнейшем будем именовать набором.

Пример:

```
script MDB,”d:\\base.mdb”,"Cel","Celsius","12"
```

То есть выполнить сценарий из набора, расположенного в базе данных **d:\\base.mdb**. Найти в наборе (базе) модуль с именем **Cel**, а в самом модуле найти функцию **Celsius**, передать ей в качестве параметра число **12** и выполнить. Естественно команда может выполняться сразу или с помощью подсказок.

Для получения возвращаемых значений необходимо воспользоваться функцией. Формат функции:

VARIANT **FMdbScript** (STRING *<FileMdb>*, STRING *<ScriptName>*, STRING *<FuncName>*, STRING *<Params>*)

Функция инициирует исполнение функции с именем *<FuncName>* в сценарии с именем *<ScriptName>*, находящимся в сборке сценариев в базе данных с полным именем файла *<FileMdb>*. Параметр *<Params>* – строка со значениями параметров, передаваемыми функции, разделенными запятой.

Пример:

```
=FMdbScript("d:\\Sz.mdb",”Cel”,"Celsius","12")
 -11.111
=FMdbScript("d:\\Sz.mdb",”Lec”,"Func1","12,15,23")
 152
```

Хранение сценариев в базе данных предпочтительнее еще и по той причине, что в базе можно хранить для каждого именованного модуля имя стартовой функции и параметры, используемые по умолчанию. В ряде случаев, это упрощает работу. Особенно в процессе отладки, когда нет необходимости постоянно вводить различные имена функций или параметров.

Заметим, что при вызове команды или функции при работе с базой нет необходимости указывать язык сценария. В самой базе для каждого модуля храниться его языковый тип. Другими словами в наборе (базе) могут храниться модули с различным языковым типом. Остальные типы хранения сценариев таким свойством не обладают.

Использование сценариев, собранных в проект

Проект – это способ организовать тексты программ. Очень часто многие функции повторяются. Например, функции работы с файловой системой хотелось бы написать один раз и затем использовать во всех сценариях. Именно для этого и предназначено понятие проекта. При его создании указываются все модули, необходимые для проекта. Модули могут находиться в разных файлах, в разных наборах. Единственное требование – одинаковый язык программирования. Это позволит исключить расточительное копирование уже написанных функций из одного модуля в другой и формировать библиотеки модулей и впоследствии использовать их по мере необходимости.

После того, как проект окончательно оформлен и отлажен, формируется промежуточный файл (сжатый), содержащий все необходимые функции для работы.

Файл проекта имеет расширение **KTP** и служит только для описательных целей. Результирующий собранный файл имеет расширение **KZP** и именно он готов к употреблению в макрокомандах.

Script PRJ *<FileKzp>*, [*<FuncName>*, [*<Params>*]]

Команда вызывает на выполнение функцию с именем *<FuncName>* в проекте сценариев в файле с именем *<FileKzp>*. *<Params>* – текстовая строка с набором переменных, разделенных запятой, передаваемых в виде параметров. В принципе имя функции *<FuncName>* и строку параметров *<Params>* можно не указывать, поскольку эти параметры задаются в самом проекте. Возможность указания сделана для обеспечения гибкости.

Пример:

```
script PRJ,”d:\\base.kzp”,"Celsius","12"
script PRJ "D:\\Comp.kzp", "", ""        // Используются умолчания
script PRJ "D:\\Comp.kzp", "Comp2", ""   //  Умолчания игнорируются
```

Естественно команда может выполняться сразу или с помощью подсказок.

Для получения возвращаемых значений необходимо воспользоваться функцией. Формат функции:

VARIANT **FPrjScript** (STRING *<FileKzp>*, STRING *<FuncName>*,

STRING <*Params*>)

Функция инициирует исполнение функции с именем <*FuncName*>, находящимся в проекте сценариев с полным именем файла <*FileKzp*>. Параметр <*Params*> – строка со значениями параметров, передаваемыми функции, разделенными запятой. При описании проекта допускается указывать функцию – точку входа. В этом случае, если требуется вызвать функцию, описанную в проекте, как точку входа, вместо имени функции допускается указывать пустую строку. Однако делать этого не рекомендуется для обеспечения гибкости проекта сценариев.

Пример:

```
=FPrjScript("d:\\Sz.kzp,"Celsius","12")
 -11.111
=FPrjScript("d:\\Sz.kzp","Func1","12,15,23")
 152
```

Динамические сценарии

Поскольку сценарий - это не что иное, как текст, передаваемый интерпретатору, то имеется возможность сформировать текст сценария непосредственно в макрокоманде и выполнить его. Единственное, что надо указать - на каком языке написана программа.

Для этого введена функция:

VARIANT **ftxtscript** ("Jscript"|"VBScript", STRING <*FuncName*>, STRING <*Params*>, STRING <*ProgText*>)

Функция инициирует исполнение сценария на языке **VisualBasic** или **JavaScript**, текст которого записан в строке .<*ProgText*> "Jscript" или "VBScript" – указание типа языка (**JavaScript** или **VisualBasic**). <*FuncName*> - имя функции, являющейся точкой входа для работы. Параметр <*Params*> содержит список параметров, передаваемых в функцию, разделенных запятыми. Если в функцию параметры передаваться не должны, то параметр <*Params*> должен содержать пустую строку. Параметр <*ProgText*> – собственно текст программы.

Пример:

```
// сформируем текст программы
szText="function YouFunc(){var s; s = \"Тут\"; for(var i = 0; ";
szText=SzText+"i < 10; i++) s += i; return(s);}";
// Выполним её
aaa=ftxtscript("JScript","YouFunc","",szText);
=aaa;
```

«Тут0123456789»

Эта функция позволяет производить те же действия, какие принято применять в структуре **HTML**, когда надо внедрить дополнительную обработку с помощью других языковых средств.

Передача параметров

При использовании функций, необходимо иметь в виду, что языки программирования, предназначенные для написания сценариев, не связаны с системой программирования **КЗ**. В таких языках, как правило, используются типы данных типа **VARIANT**. Отсюда возникает необходимость преобразования типов.

В сценарий в качестве параметра передается строка. В строке параметры разделены запятыми. Функция или модуль преобразует параметры в соответствии со своими потребностями.

Например:

```
Function Celsius(fDegrees)
  Celsius = CDate(Celsius) ‘ Преобразование к дате
  Celsius = "Май 19, 1962"
  Celsius = Now

  Celsius = CCur(2)
  Celsius = 12.10

  Celsius = “Тектовая строка”
  Celsius = 123.45

End Function
```

Когда функция сценария возвращает какое-либо значение, то преобразование данных для их использования в **КЗ** производится по следующим правилам[16]:

Возвращаемое значение из VBScript	Тип в КЗ
Byte	double
Short	double
Long	double
Float	double
Double	double
VARIANT_BOOL	double
CY	double

DATE	Char*
BSTR	Char*

Преобразование типов

Перекрестные вызовы набора

Когда говорилось о систематизации сценариев при их хранении в базе данных, естественным образом возникает вопрос: В случае, когда все функции находятся в пределах одного модуля – все просто, что делать, если нужная функция расположена в тексте другого модуля и возможно в другой базе данных (наборе)?

Для обеспечения возможности таких вызовов в язык сценариев введены дополнительные функции:

K3F.ExtCall0 (*<strMdb>, <strScript>, <strFunc>*);

K3F.ExtCall1 (*<strMdb>, <strScript>, <strFunc>*, *<Par1>*);

K3F.ExtCall2 (*<strMdb>, <strScript>, <strFunc>*, *<Par1>*,*<Par2>*);

K3F.ExtCall3 (*<strMdb>, <strScript>, <strFunc>*, *<Par1>*, *<Par2>*, *<Par3>*);

K3F.ExtCall4 (*<strMdb>, <strScript>, <strFunc>*, *<Par1>*, *<Par2>*, *<Par3>*, *<Par4>*);

K3F.ExtCall5 (*<strMdb>, <strScript>, <strFunc>*, *<Par1>*, *<Par2>*, *<Par3>*, *<Par4>*, *<Par5>*);

Функции отличаются друг от друга только числом параметров. Поэтому и имена практически совпадают – с одним отличием: в названии функции участвует число, говорящее о том, сколько параметров передается при вызове.

Работу этих функций проще пояснить на примере.

Предположим, существуют две базы данных со сценариями: **Main.mdb** и **Other.mdb.** В базе **Main.mdb** есть два сценария: **StartModul** и **SecondModul.** В **Other.mdb** есть тоже сценарий **ThirdModul**.

В каждом из сценариев есть свои функции. **StartModul** в базе **Main.mdb** содержит:

```
Sub StartFunction() ‘ Стартовая функция
Dim Val
  Val = "Привет"
  K3F.Display(Val)
  SecondFunction(Val)’ Вызов локальной функции
End Sub

Sub SecondFunction(Parametr)
Dim Val
  Val = Parametr
  Val = Val+ "Привет"
  K3F.Display Val
  ‘Вызовем из другого сценария той же базы данных
  K3F.ExtCall1 "D:\ODEGOV\Main.mdb","SecondModul","SecondFunction",Val
  ‘Вызовем из другого сценария другой базы данных
  Val = K3F.ExtCall1 ("D:\ODEGOV\other.mdb","Third","ThirdFunction",Val)
  K3F.Display(Val)
End Sub
```

SecondModul в базе **Main.mdb:**

```
Sub SecondFunction(Parametr)
  Val = Parametr
  Val = Val + "Привет"
  K3F.Display(Val)
End Sub
```

ThirdModul в базе **Other.mdb** есть то же сценарий:

```
Function ThirdFunction(Parametr)
Dim Val
  Val = Parametr
  Val = Val+ "Привет но уже из другой MDB"
  K3F.Display Val
  ThirdFunction = Val + "С добавкой"
End Function
```

Таким образом, можно вызывать функции из произвольного места, с единственным ограничением: максимальное число передаваемых параметров не должно превышать пяти.

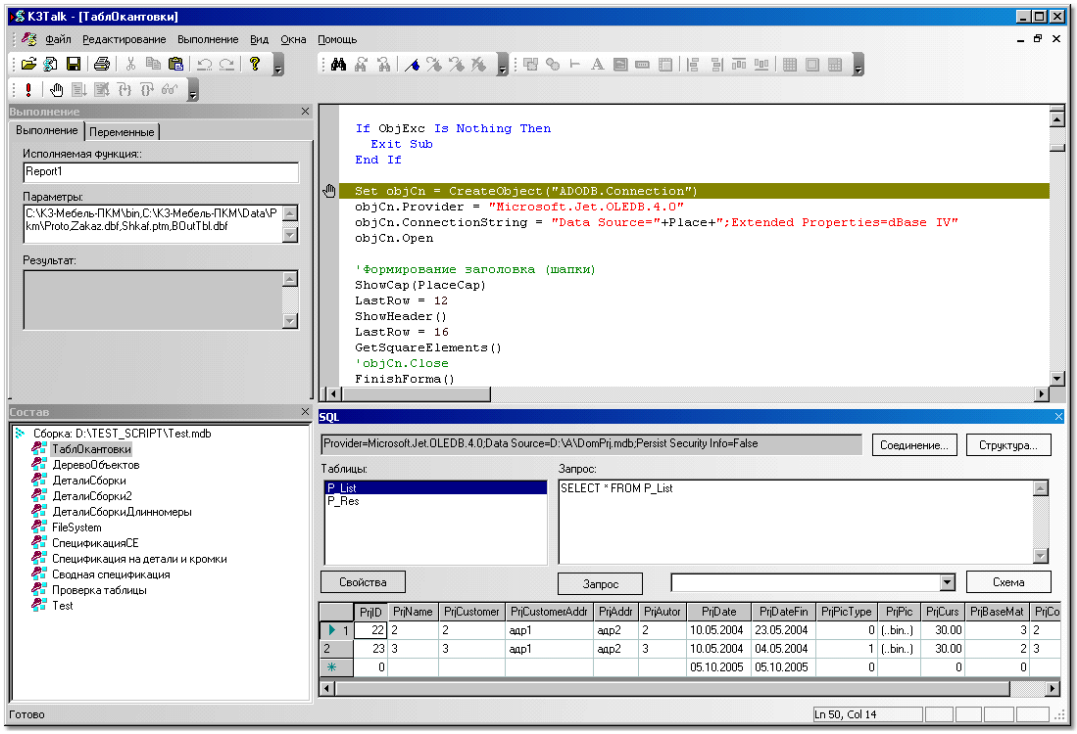
Примечание. *Эти функции применимы только к наборам.*

Внимание!!! *Данная возможность оставлена исключительно в целях обеспечения совместимости с предыдущими версиями. При работе с* ***K3Talk*** *вызов этих функций будет приводить к ошибке. Вместо этих функций предпочтительнее использовать создание проектов.*

Редактор сценариев K3Talk

Для облегчения работы, в составе комплекса **К3** поставляется редактор, призванный облегчить процесс создания сценариев. Как указывалось в самом начале, сценарии оказывают неоценимую помощь при работе с приложениями, поддерживающими автоматизацию.

Одним из наиболее мощных объектов автоматизации является **ADO** (**ActiveX Data Object**) – средство обмена данными с различными источниками баз данных. Поэтому особый акцент в приложении **K3Talk** сделан на обеспечении возможности во время отладки сценариев, работающих с базами данных, осуществлять разнообразные запросы к базам. Это позволит явно сравнивать результаты запросов с результатами работы сценария.



Внешний вид приложения после запуска:

Окно приложения 18 разделено на четыре части.

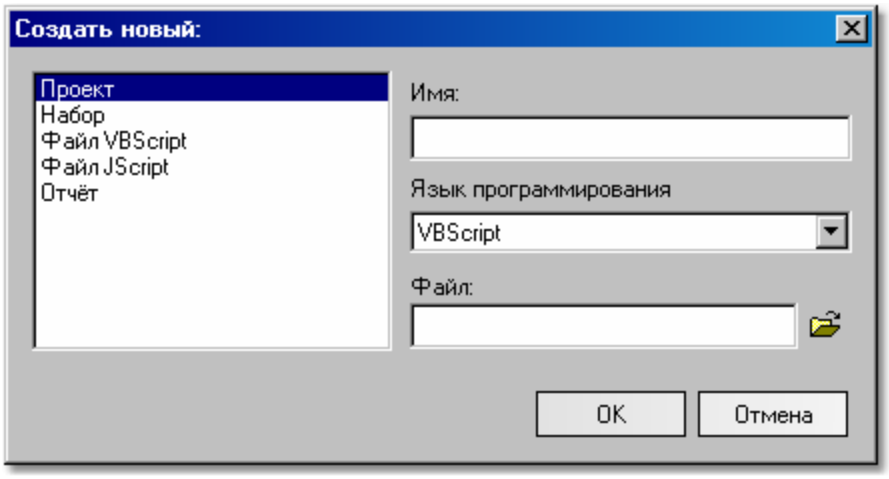
1. Окно редактирования;

2. Окно выполнения (параметры запуска и просмотр переменных);
3. Окно тестирования запросов к базам данных;
4. Структура проекта (набора).

Общие принципы работы с редактором и отладчиком мало отличаются от общепринятых способов работы с аналогичными приложениями.

Создание модулей, наборов, проектов

Выполним в **K3Talk** команду меню "**Файл/Новый**". На экране появится диалоговое окно 19:



Файл/Новый

В диалоговом окне предлагается создать либо проект (файл с расширением **KTP**), либо набор (база типа **MDB**), либо отдельные файлы.

Создание модуля в файле

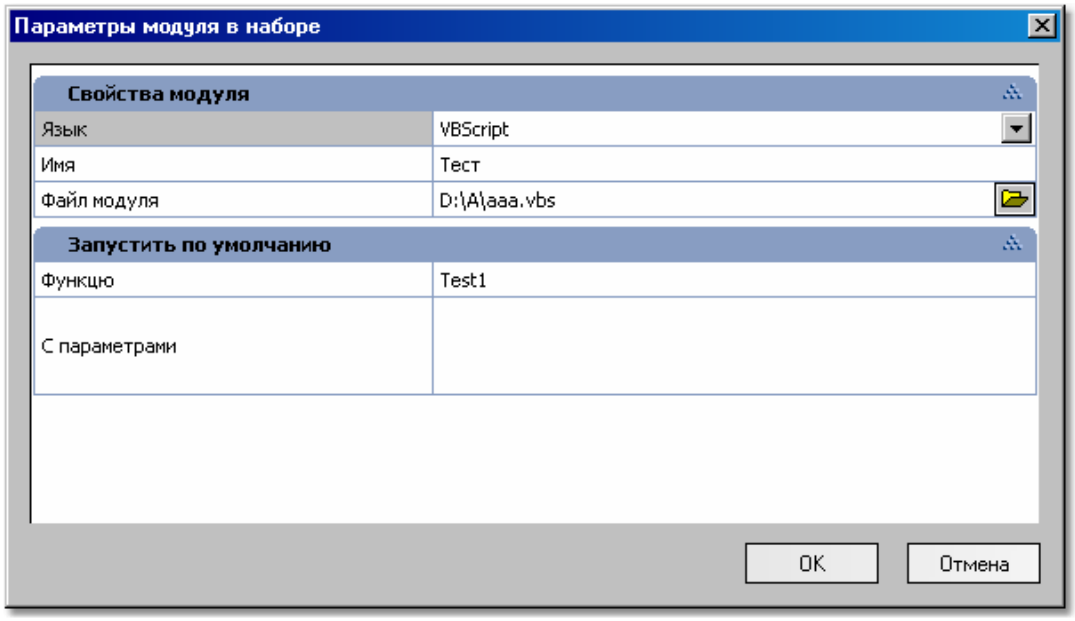
Если выбирается файл, он создаётся и в него автоматически вставляется функция. Например, вида:

```
function YouFunc()
{
  return
}
```

После чего можно наполнять сценарий осмысленными действиями и отлаживать его.

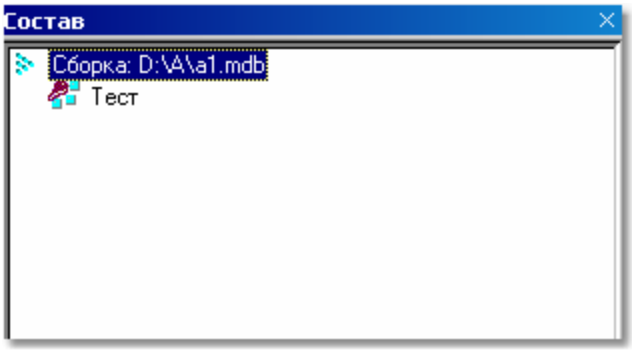
Создание набора модулей

Если выбирается набор, то автоматически создаётся пустая база данных. Для ее наполнения нужно перейти в окно “**Состав**” и щелкнуть правой кнопкой мыши на корневом элементе. Здесь же есть возможность завершить работу с набором и есть возможность добавить в набор модуль. Выберем “**Добавить**”. Появится диалоговое окно 20:



Добавление модуля в набор

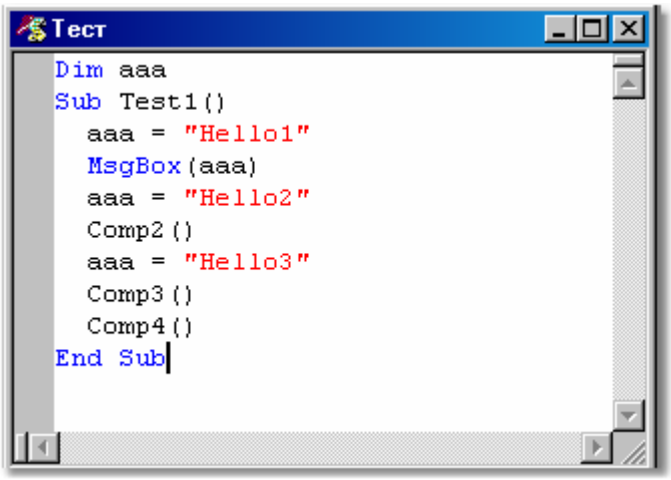
В нём необходимо задать тип языка для вставляемого модуля. Дать ему имя, поскольку из файла он будет перенесён в базу. Затем необходимо указать функцию, вызываемую по-умолчанию, и параметры, используемые по-умолчанию. После того, как все необходимые действия будут выполнены и будет нажата **OK**, в окне “**Состав**” появится новая запись 21:



Модуль в набор добавлен

Это означает, что в набор добавлен новый модуль.

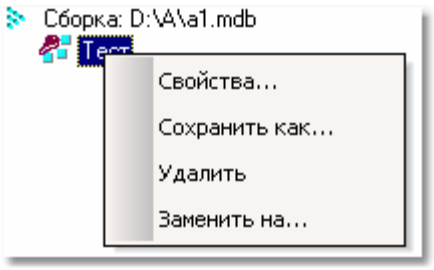
Что-бы открыть его для редактирования и отладки достаточно дважды щелкнуть левой кнопкой мыши на имени модуля. На экране появится окно редактирования:



Окно редактирования

Теперь можно приступать к отладке и при необходимости редактировать текст сценария. Все изменения в тексте сценария будут сохраняться в наборе (базе)

Для любого модуля в наборе существуют дополнительные операции. Выбор операции производиться с помощью меню 22, возникающего при нажатии правой кнопкой мыши на модуле набора:



Дополнительные операции

Свойства – выводит на экран параметры модуля, заданные по умолчанию. Часть свойств можно изменять.

Сохранить как – модуль набора извлекается и сохраняется в указанном файле.

Удалить – удаляет модуль из набора.

Заменить на – текст модуля набора заменяется на текст из указанного файла.

После того, как работу с набором завершена, его необходимо закрыть. Закрытие набора приводит также к закрытию окон редактирования, модули которых принадлежат набору.

Создание проекта

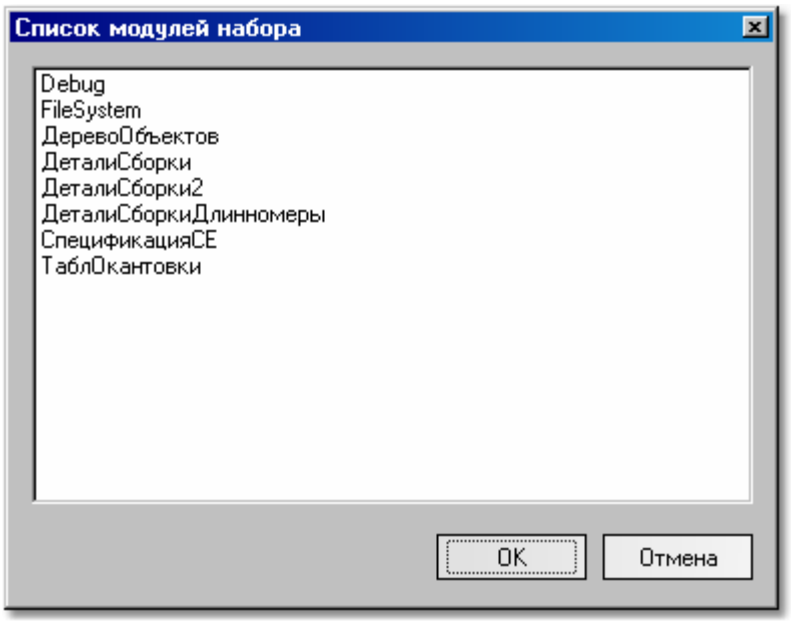
Если выбрано создание проекта, то требуется задать его имя, используемый язык и местоположение. В дальнейшем, в зависимости от выбранного языка будут предлагаться для добавления только модули соответствующего языкового типа.

Добавление модулей в проект производиться с помощью команды меню “**Добавить...**”, появляющегося при нажатии правой кнопкой мыши на корневом элементе окна “**Состав**”.

Добавлять модули в проект можно из двух источников:

- из файлов
- из наборов

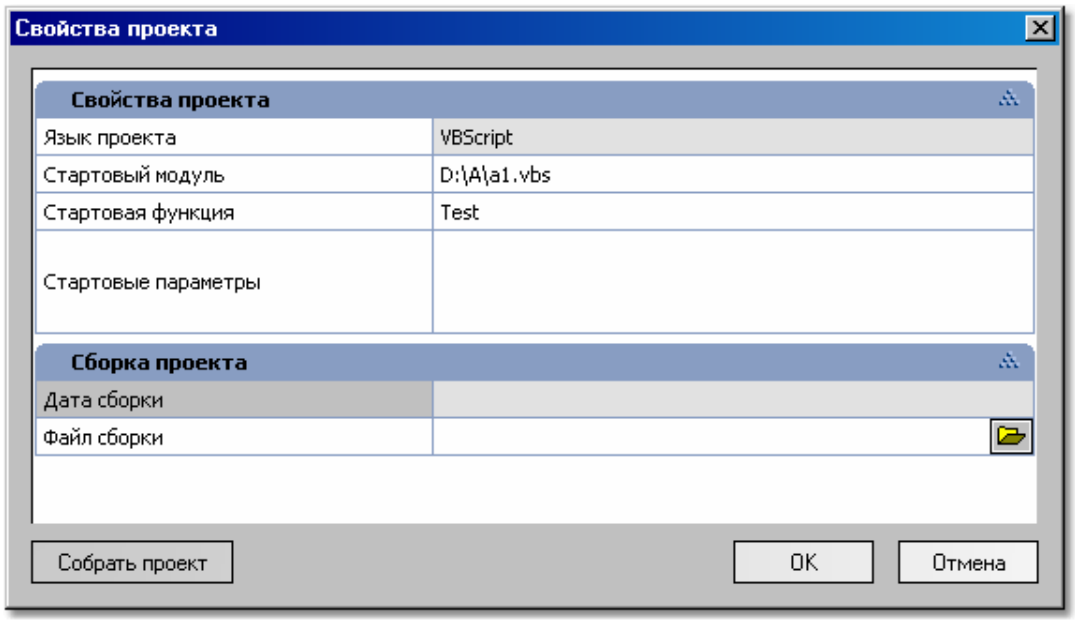
Для добавления файла в проект используется стандартный диалоговое окно открытия файла. В случае, если вставляется модуль из набора, предварительно предлагается список 23 имен модулей, содержащихся в наборе:



Список модулей набора

В списке будут присутствовать только те модули, которые соответствуют языковой совместимости. Чтобы открыть один из модулей проекта для редактирования и отладки, достаточно дважды щелкнуть левой кнопкой мыши на имени модуля. На экране появится окно редактирования.

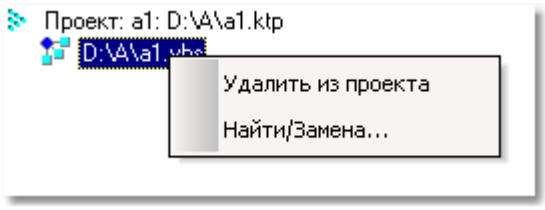
Для того, чтобы проект знал точку входа, необходимо задать его параметры по-умолчанию. Параметры задаются командой меню “**Свойства проекта...**”



Свойства проекта

В этом же диалоговом окне расположена кнопка, формирующая собранный проект для использования в командах **КЗ** – “**Собрать проект**”.

При нажатии правой кнопкой мыши на каком-либо модуле проекта, появляется меню:

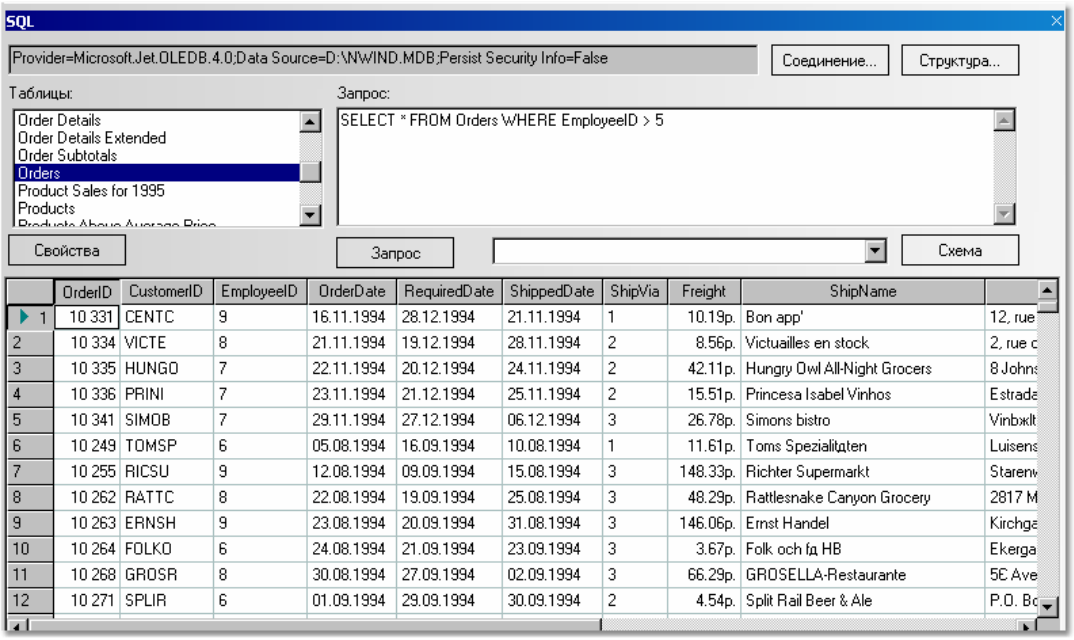


Меню модулей проекта

- **Удалить из проекта** – удаляет выбранный модуль из проекта
- **Найти/Замена** – заменяет выбранный модуль на новый. Процедура замены похожа на вставку нового модуля.

Одновременное тестирование данных базы данных

Очень часто работа сценариев связана с формированием разнообразных запросов к базам данных. В этом случае, было бы удобно формировать запрос и получить результат без прохождения отладчиком по циклам. Для этого служит **панель SQL** 25.



Панель SQL

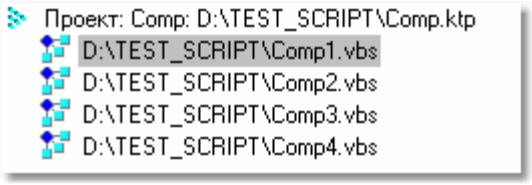
На рисунке 25 видно, что:

1. Выполнялось соединение с базой данных **NWIND.MDB** (Кнопка “**Соединение**”);
2. Сформировался запрос (текстовое поле “**Запрос**”);
3. Выполнялся запрос (кнопка ”**Запрос**”);
4. Результата выведен в таблицу.

Эти действия можно производить в момент отладки сценариев, что существенно увеличивает наглядность получаемых результатов.

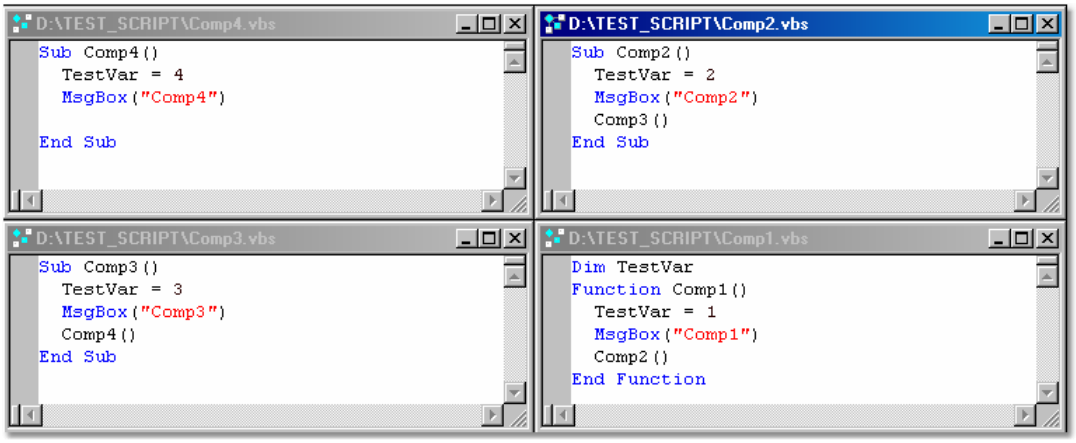
Отладка

Процедура отладки начинается с выбора модуля, набора или проекта. Загрузим проект 26. Например, состоящий из модулей:



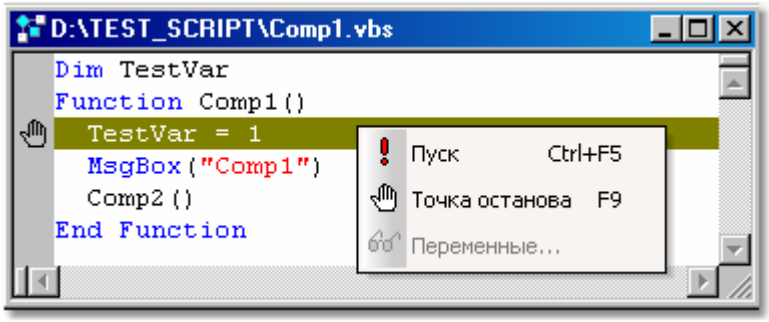
Пример проекта

Откроем все модули 26:



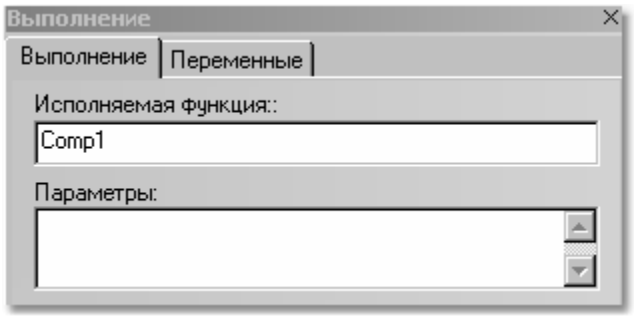
Модули проекта

Точкой входа будет функция **Comp1** из модуля **Comp1.vbs**. Есть последовательность вызовов функций из одного модуля в другой. Есть глобальная переменная `TestVar`. Поставим точку останова 26 (**F9**)



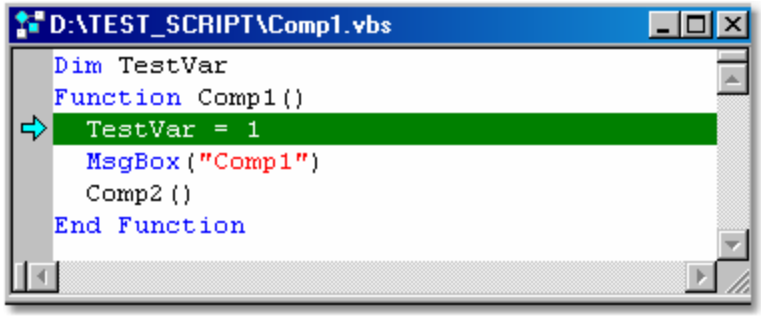
Точка останова

Запустим проект на выполнение (**Ctrl+F5**). При этом в окне “**Выполнение**” должна быть указана стартовая функция. Эта функция не имеет параметров:



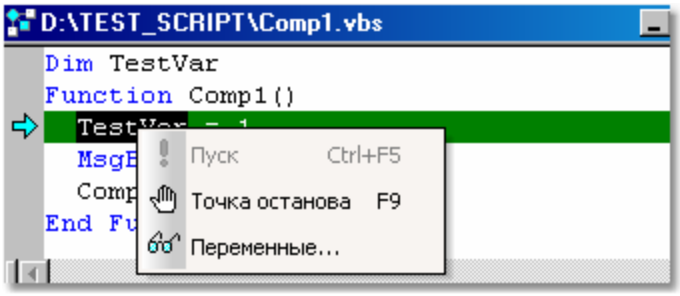
Окно "Выполнение"

После запуска сценарий будет остановлен в заданном месте:



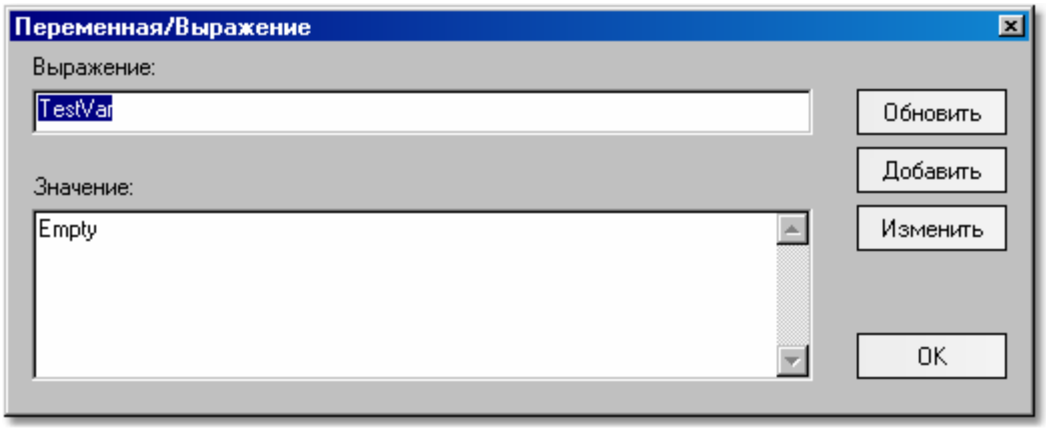
Останов сценария

Следующий шаг должен привести к заполнению переменной. Для того, чтобы наблюдать за ней, выберем её двойным щелчком левой кнопки мыши и выполним команду меню “**Переменные...**”



Переменные

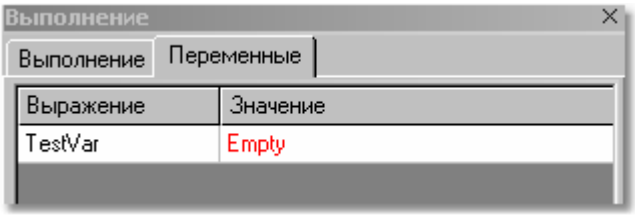
Появится диалоговое окно:



Переменные и выражения

Пока переменная не инициализирована. В процессе отладки переменным можно присваивать другие значения. Если же необходимо постоянно наблюдать за ней – выберем кнопку “**Добавить** ” и закроем диалоговое окно.

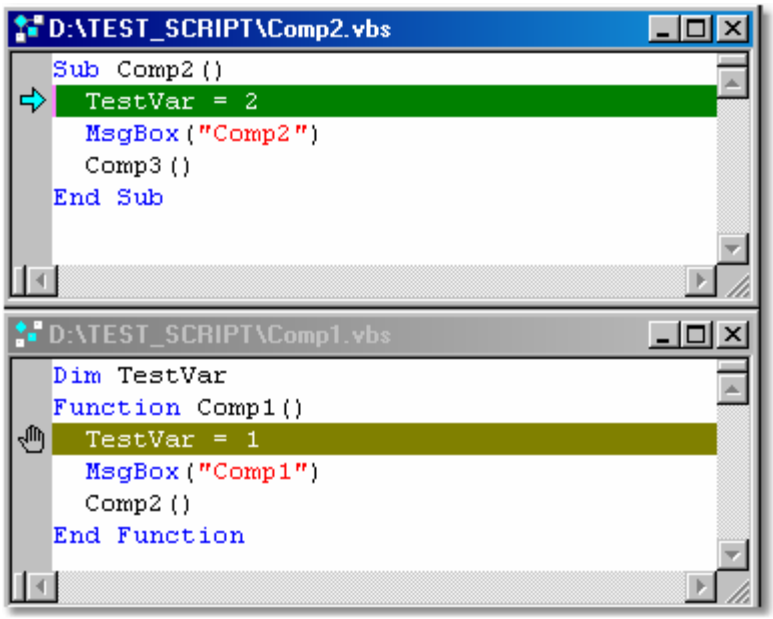
Переключимся в окне “**Выполнение**” на закладку “**Переменные** 28”.



Переменые/Значения

В списке переменных мы увидим нашу переменную с текущим значением.

Вернёмся к отладке. Нажмем **F10** – выполнить один шаг. Увидим, что переменной будет присвоено значение равное **1** (окно “ **Выполнение**”). Следующее нажатие **F10** – выполнить еще один шаг, приведет к появлению сообщения. И теперь мы стоим на имени функции, находящейся в другом модуле. Для входа в функцию используется **F11**. После её нажатия мы окажемся в другой функции другого модуля 29.



Переход в функцию другого модуля

Таким способом можно пройти по шагам весь проект.

Примеры работы

Как ранее упоминалось, в системе **К3** использование сценариев в основном предназначено для обеспечения связи с программами, поддерживающих автоматизацию.

Приведем несколько примеров.

```
Function Test()
  Set obj = CreateObject("Access.Application")
  If Not obj Is Nothing Then
    ' Создадим новый документ – это стандартная nwind.mdb
    obj.OpenCurrentDatabase "d:\nwind.mdb"
    obj.Visible = True
    obj.DoCmd.OpenForm "Employees"
  End If
End Function
```

В функции сценария мы создаем объект типа **Access** и заставляем вывести форму **Employees**.

Аналогичным образом можно открыть **Excel** и заставить выполнить некоторые действия в пределах таблицы.

```
Sub Test()
  Dim obj
  K3F.Display ("asda")
  Set obj = CreateObject("Excel.Application")

  If Not obj Is Nothing Then
    obj.Visible = True
    obj.Workbooks.Add
    obj.Range("A1").Value = 1234
    obj.Range("A2").Value = 1235
    obj.Range("A3").Value = 1236
    obj.Range("D1").Value = "Привет!"
    obj.ActiveWorkBook.SaveAs("D:\Serge.xls")
    obj.Quit
    Set obj = Nothing
  End If
End Sub
```

В следующем примере мы откроем базу данных в формате **dbf** и пробежимся по записям.

```
Sub Test()
  Set objCn = CreateObject("ADODB.Connection")
  Set objRs = CreateObject("ADODB.Recordset")

  objCn.Provider = "Microsoft.Jet.OLEDB.4.0"
  objCn.ConnectionString = "Data Source=D:\z;Extended Properties=dBase 5.0"
  objCn.Open

  objRs.CursorType = 1'Это adOpenKeySet
  objRs.Open "SELECT * FROM brevno.dbf" ,objCn

  objRs.MoveFirst()

  Dim tt
  Do While Not objRs.EOF
    objRs.MoveNext()
    tt = tt + 1
  Loop
  MsgBox(tt)

  objRs.MoveLast
  tt = objRs.RecordCount
  MsgBox(tt)

  tt =  objRs.Fields.Count',objCn
  MsgBox(tt)
  'objRs'.Fields'(Count)',objCn

  objRs.Close
  objCn.Close

End Sub
```

Следующий пример более сложный. В нем принимаются в качестве параметров имена баз данных и в таблицу **Excel** выводятся расчетные значения. Пример взят из рабочей версии **К3-Мебель**.

В примере показаны простые методы формирования запросов к базе данных, выборка данных, форматирования ячеек, вывод данных;

```
Dim objCn                 'Объект - соединение с базой
Dim ObjExc                'Объект Excel
Dim dbShk                 'База с параметрами шкафов
Dim dbOut                 'Выходная база
Dim TableColums           'Количество колонок в таблице

Sub Report1(Place,db1,db2)

  dbOut = db2
  dbShk = db1

  Set objCn = CreateObject("ADODB.Connection")
  objCn.Provider = "Microsoft.Jet.OLEDB.4.0"
  objCn.ConnectionString = "Data Source="+Place+";Extended Properties=dBase IV"
  objCn.Open

  CreateMyExcel()

  If ObjExc Is Nothing Then
    Exit Sub
  End If

  ShowHeader()
  GetSquareElements()

  objCn.Close
End Sub

'Создадим приложение Excel
Sub CreateMyExcel()
  Set ObjExc = CreateObject("Excel.Application")
  If Not ObjExc Is Nothing Then
    ' Создадим новый документ
    ObjExc.Visible = True
    ObjExc.Workbooks.Add
  End If
End Sub

' Подготовка заголовка
Sub ShowHeader()
  objExc.Range("A1").Value = "ЗАКАЗ - НАРЯД № ____"
  objExc.Range("A2").Value = "Спецификация на основные детали"
  Set Tab = objExc.Range("A1:F1")'.Select
  With Tab
.HorizontalAlignment = 3
.VerticalAlignment = 2
 .WrapText = True
.Orientation = 0
.AddIndent = False
.ShrinkToFit = False
.MergeCells = True
  End With
```

```
  objExc.Range("A1:F1").Font.Bold = True
Set Tab = objExc.Range("A2:F3")'.Select
  With Tab
.HorizontalAlignment = 3
.VerticalAlignment = 2
.WrapText = True
.Orientation = 0
.AddIndent = False
.ShrinkToFit = False
 .MergeCells = True
  End With

Set Tab = objExc.Range("A4:F4")'.Select
  With Tab
.HorizontalAlignment = 3
.VerticalAlignment = 2
 .WrapText = True
 .Orientation = 0
.AddIndent = False
.ShrinkToFit = False
.MergeCells = False
  End With

  Set Tab = objExc.Range("B:B")'.Select
  With Tab
.HorizontalAlignment = 3
.VerticalAlignment = 2
.WrapText = True
.Orientation = 0
.AddIndent = False
.ShrinkToFit = False
End With

  objExc.Range("A2:F3").Font.Bold = True
  objExc.Rows("1:1").RowHeight = 40.0

  objExc.Columns("A:A").ColumnWidth = 15
  objExc.Columns("B:B").ColumnWidth = 15
  objExc.Columns("C:C").ColumnWidth = 20
  objExc.Columns("D:D").ColumnWidth = 15
  objExc.Columns("E:E").ColumnWidth = 5
  objExc.Columns("F:F").ColumnWidth = 10

  objExc.Range("A4").Value = "Наименование"
  objExc.Range("B4").Value = "Номера"
  objExc.Range("C4").Value = "Элемент"
  objExc.Range("D4").Value = "Размеры"
  objExc.Range("E4").Value = "Кол-во"
  objExc.Range("F4").Value = "Пл/Длина"
  TableColums = 6
  Rows = 1
  MyRamka ("A4:F4")
```

```
End Sub

' Функция рисования рамки
Sub MyRamka(Area)
Dim Rows
  Set Sel = objExc.Range(Area)'.Select
  Rows = Sel.Count
  With Sel.Borders(7)'(xlEdgeLeft)
    .LineStyle = 1'xlContinuous
    .Weight= 3
  End With
  With Sel.Borders(8)'(xlEdgeTop)
    .LineStyle = 1'xlContinuous
    .Weight= 3
  End With
  With Sel.Borders(10)'(xlEdgeRight)
    .LineStyle = 1'xlContinuous
    .Weight= 3
  End With
  With Sel.Borders(9)'(xlEdgeBottom)
    .LineStyle = 1'xlContinuous
    .Weight= 3
  End With
  With Sel.Borders(11)'(xlInsideVertical)
    .LineStyle = 1'xlContinuous
  End With
  If Rows > TableColums Then
    With Sel.Borders(12)'(xlInsideHorizontal)
      .LineStyle = 1'xlContinuous
     End With
  End If
End Sub


Function IsNotNumsInStr(Str,Search)
Dim Pos
  Pos = Instr(1,Str,Search)
  If(Pos = 0) Then
    IsNotNumsInStr = True
  Else
    IsNotNumsInStr = False
  End If
End Function


'Вывод элементов по площади
Sub GetSquareElements()

Dim   Rows1,Rows2,Rows3
Dim   SqlStr,Str,StrNN
Dim   Par1,Par2,Par3,Par4
Dim   I1,I2,I3,I4
```

```
Dim   FactSize

Dim   MatName
Dim   DetName
Dim   DetNum
Dim   LastRow
Dim   Sqare
Dim   SumSqare
Dim   Nums




  LastRow = 5
  'Сразу создадим, что нам нужно
  Set objRs1 = CreateObject("ADODB.Recordset")
  objRs1.CursorType = 1'Это adOpenKeySet

  SqlStr = "SELECT DISTINCT PRICEID,COD,MATNAME FROM " + dbOut + "," + dbShk
  " WHERE UNITS = 2 AND PRICEID = COD"

  objRs1.Open SqlStr,objCn
  objRs1.MoveFirst
  objRs1.MoveLast
  objRs1.MoveFirst
  Rows1 = objRs1.RecordCount

  For I1 = 1 To Rows1
    Par1 = objRs1.Fields(0)
    MatName = objRs1.Fields(2)
    objExc.Range("A"&LastRow).Value = MatName
    'K3F.Display(MatName)
    Set objRs2 = CreateObject("ADODB.Recordset")
    objRs2.CursorType = 1'Это adOpenKeySet
    SqlStr = "SELECT NAME, XUNIT, YUNIT FROM " + dbOut + _
                    " WHERE PRICEID = " &Par1& " GROUP BY NAME,XUNIT,YUNIT"
    objRs2.Open SqlStr,objCn
    objRs2.MoveFirst
    objRs2.MoveLast
    objRs2.MoveFirst
    Rows2 = objRs2.RecordCount
    DetName =  objRs2.Fields(0)
    DetNum = Rows2
    SumSqare = 0
    For I2 = 1 To Rows2

        Par2 = objRs2.Fields(0)
        Par3 = Abs(objRs2.Fields(1))
        Par4 = Abs(objRs2.Fields(2))

        objExc.Range("C"&(LastRow+I2-1)).Value = Par2'DetName
        Str = Par4&" x "&Par3
```

```
        objExc.Range("D"&(LastRow+I2-1)).Value = Str'DetName

        Set objRs3 = CreateObject("ADODB.Recordset")
        objRs3.CursorType = 1'Это adOpenKeySet

        ' Посколку формат дробный - нужно преобразование для запроса
        Par3 = Replace(Par3,",",".")
        Par4 = Replace(Par4,",",".")

        SqlStr = "SELECT NUMDET, XUNIT, YUNIT FROM " + dbOut + _
            " WHERE PRICEID = " &Par1& _
            " AND NAME = "+"'"+Par2+"'"+"_
            AND XUNIT = "&Par3&" AND YUNIT = "&Par4
        objRs3.Open SqlStr,objCn
        objRs3.MoveFirst
        objRs3.MoveLast
        Rows3 = objRs3.RecordCount
        objRs3.MoveFirst

        objExc.Range("E"&(LastRow+I2-1)).Value = Rows3
        Sqare = objRs3.Fields(1)*objRs3.Fields(2)*Rows3
        Sqare = Round(Sqare/1000000.0,2)
        SumSqare = SumSqare + Sqare

        Nums = ""
        For I3 = 1 To Rows3
          FactSize = objRs3.Fields(0).ActualSize
' Собственно это проверка на действительность поля - его могут не заполнить
          If FactSize > 0 Then
            StrNN = StrNN&"  "&Cstr(objRs3.Fields(0))
            If(IsNotNumsInStr(Nums,StrNN)) Then
              Nums = Nums&Cstr(objRs3.Fields(0))&" "
            End If
          Else
            ' Поищем такой уже там в строке
            If(IsNotNumsInStr(Nums," ?")) Then
              Nums = Nums&" ?"
            End If
          End If

          objRs3.MoveNext
        Next
        objExc.Range("B"&(LastRow+I2-1)).Value = Nums
        objExc.Range("F"&(LastRow+I2-1)).Value = Sqare

        objRs3.Close
        objRs2.MoveNext
    Next
'====================================================================
    'Раскраска материала
    Set Tab = objExc.Range("A"&LastRow&":"&"A"&(LastRow+DetNum))'.Select
    With Tab
```

```
        .HorizontalAlignment = 1
       .VerticalAlignment = 2
       .WrapText = True
       .Orientation = 0
       .AddIndent = False
        .ShrinkToFit = False
        .MergeCells = True
    End With

    Set Tab = objExc.Range("B"&(LastRow+DetNum)&":"&"F"&(LastRow+DetNum))'.S
    With Tab
        .HorizontalAlignment = 4
       .VerticalAlignment = 2
       .WrapText = True
       .Orientation = 0
       .AddIndent = False
        .ShrinkToFit = False
        .MergeCells = True
    End With
    Str = "Итого "&MatName&":    "&SumSqare
    objExc.Range("B"&(LastRow+DetNum)).Value = Str

    MyRamka ("A"&LastRow&":"&"F"&(LastRow+DetNum))
'=====================================================================
    'K3F.Display(MatName)
    'K3F.Display(DetNum)
    LastRow = LastRow+DetNum+1
    objRs2.Close
    objRs1.MoveNext
  Next
  objRs1.Close
End Sub
```

Результатом работы сценария будет следующая таблица:

ЗАКАЗ - НАРЯД № _____					
Спецификация на основные детали					
Наименование	Номера	Элемент	Размеры	Кол-во	Пл/Длина

ДСП 16	0 0 0 0 0 0	Боковина ящика	450 x 158	6	0,43
	10	Полка	499,8 x 500	1	0,25
	10 10 10	Полка	500 x 500	3	0,75
	7	Полка	500 x 718	1	0,36
	8	Полка	500 x 1234	1	0,62
	3	Полка	600 x 1968	1	1,18
	5	Полка	600 x 2000	1	1,2
	4	Стенка	1968 x 50	1	0,1
	0 0 0 0 0 0	Стенка ящика	443 x 158	6	0,42
	6	Стойка	500 x 1266	1	0,63
	9	Стойка	500 x 1618	1	0,81
	1 2	Стойка	600 x 1684	2	2,02
	Итого ДСП 16: 8,77				
ДВП Белая	0 0 0	Дно ящика	434 x 459	3	0,6
	Итого ДВП Белая: 0,6				
ДСП 16 Ольха	0 0 0	Полотно фасада	495 x 174	3	0,26
	Итого ДСП 16 Ольха: 0,26				
ДСП 8 Вишня	0 0 0	Запол.двери	631 x 1512	3	2,86
	Итого ДСП 8 Вишня: 2,86				

В качестве примера можно привести текст сценария, использующего **Jscript**.

```
function SampleObj()
{
var  ExcelApp;
var  WshShell;
var  Result;
var  i;
var  s;

  ExcelApp = new ActiveXObject("Excel.Application");
  ExcelApp.Visible = true;
  ExcelApp.Workbooks.Add;
  for(i  =  1;  i < 20;  i++)
  {
    s = "A" + i;
    ExcelApp.Range(s).Value = s;
  }

  WshShell = new ActiveXObject("WScript.Shell");
  Result = WshShell.Popup("Привет из сценария",50000,"JScript",65);
  WshShell.Run("Calc.exe");

  delete (ExcelApp);
  delete (WshShell);

}
```

Видно насколько похож синтаксис языка на синтаксис **C**.

В этом коротком примере:

1. Создаётся объект **Excel.Application**;
2. В него добавляется новая рабочая страница;
3. Заполняются ячейки;
4. Создаётся объект “**Оболочка**”;
5. С помощью оболочки выдаётся сообщение и запускается калькулятор;
6. Созданные объекты уничтожаются.

Использование макросов из MSOffice

В предыдущих примерах показано, как использовать методы и свойства объектов **MSOffice**. Среди них приемы форматирования ячеек. Откуда взять информацию об этом?

1. Это конечно документация на приложение, используемое как объект.
2. Использование макросов в качестве подсказок.

MSOffice обладает инструментом автоматической генерацией макросов по последовательности действий пользователя. Например, в **Excel** можно выполнить команду меню **Сервис|Макрос|Начать запись...**, выполнить некоторый набор действий и запомнить макрос. Затем в режиме редактирования просмотреть, что же было сформировано. Текст макроса будет великолепной подсказкой. Тем более, что он будет сформирован на языке **VisualBasic for Application** – близком родственнике **VisualBasic Script**.

Единственное неудобство заключается в том, что макросы используют в качестве констант именованные значения. Например, при форматировании ячейки в **Excel** макрос будет использовать:

```
.HorizontalAlignment = xlCenter
.VerticalAlignment = xlBottom
```

`xlCenter` и `xlBottom` – это внутренние именованные константы **MSOffice**. **VBScript** не может их использовать. Вместо них необходимо подставить явные числовые значения.

```
.HorizontalAlignment = 2
.VerticalAlignment = 3
```

Предметный указатель

- А -

- В -

- Д -

- М -

- О -

- П -

- С -

- Т -

- Ф -