# СОДЕРЖАНИЕ

# СОДЕРЖАНИЕ

Стр.

# 12 ЯЗЫК СЦЕНАРИЕВ КРУГ-VBScript

## 12.1 Общие положения

Программа предназначена для использования языка **VBScript** в SCADA КРУГ-2000. Использование скриптов позволяет более гибко управлять значениями переменных, а также предоставляет широкий спектр возможностей для создания динамичных проектов, с использованием анимации и звуковых эффектов.

**VBScript** – это простой язык скриптов, который тем не менее, поддерживает такие синтаксические конструкции как процедуры, функции, различные арифметические операции, классы, циклы и другие, а также позволяет объявлять и использовать переменные и константы.

Главное достоинство **VBScript** – это то, что его можно встраивать в различные приложения, а также добавлять в него свои функции, конструкции и объекты.

Таким образом, язык скриптов, используемый в КРУГ-2000, базируется на языке **VBScript**, но его функциональность расширена, за счет использования встроенных функций, объектов, библиотек и переменных, используемых для связи с базой данных.

Скрипты хранятся в текстовых файлах с расширением **ks;** библиотеки скриптов – в файлах с расширением **kl**.

Предусмотрена возможность, отключить использование скриптов – при запуске **ГД** или **ГИ** следует указать параметр **/NoScript**.

Использование параметра **/UseEditor** в командной строке при запуске Станции оператора (Графического интерфейса) инициирует вывод в окно редактора информации времени выполнения, сообщений об ошибках, а также формирование лога.

## 12.2 Типы данных

В **VBScript** имеется единственный тип данных, называемый ***Variant***.

**Variant** – это специальный тип данных, который может содержать различную информацию, в зависимости от ситуации.

Поскольку **Variant** единственный тип данных, используемый в **VBScript**, все функции **VBScript** возвращают значения этого типа.

**Variant** может содержать как числа, так и строки. Тип ведет себя как число, когда вы используете его в числовом контексте, и как строка когда вы используете его в контексте строки. Числовые значения в свою очередь, могут содержать в себе различные типы данных, например, дату или время, логические значения и так далее. Все эти типы данных называются подтипами **Variant.** Следующая таблица показывает подтипы данных, которые может содержать **Variant.**

Таблица 12.2.1

| Подтип | Описание |
|---|---|
| **Empty** | **Variant** не инициализирован. Значение будет равно 0, если это числовая переменная, или пустая строка (длина = 0), если переменная строковая. |
| **Null** | **Variant** не содержит никаких типов данных |
| **Boolean** | Значения: **True**, эквивалентное -1, или **False**, эквивалентное 0. |

| Подтип | Описание |
|---|---|
| **Byte** | Целое в интервале от 0 до 255. |
| **Integer** | Целое в интервале от –32768 до 32767. |
| **Currency** | От -922,337,203,685,477.5808 до 922,337,203,685,477.5807. |
| **Long** | Целое в интервале от<br>-2,147,483,648 до 2,147,483,647. |
| **Single** | Число с плавающей точкой одинарной точности в диапазоне<br>-3.402823E38 до -1.401298E-45 для отрицательных значений;<br>1.401298E-45 to 3.402823E38 для положительных. |
| **Double** | Число с плавающей точкой двойной точности в диапазоне<br>-1.79769313486232E308 to -4.94065645841247E-324 для отрицательных значений;<br>4.94065645841247E-324 для положительных. |
| **Date (Time)** | Число представляющее дату между 1 января 100 г. и 31 Декабря 9999 г. |
| **String** | Строка переменной длины, которая может достигать 2 миллиарда символов. |
| **Object** | Объект. |
| **Error** | Номер ошибки. |

### 12.3 Константы

**Константа** – это неизменяемое числовое или строковое значение, имеющее предопределенное имя.

**VBScript** содержит набор встроенных констант.

Пользователь может определять свои собственные константы с помощью конструкции **Const**, например: **Const MyString = "This is my string."**

**Const MyAge = 49**

Строковые константы должны заключаться в кавычки. Это необходимо для того, чтобы отличить стоку от числового значения.

Чтобы показать, что значение является датой, оно обрамляется знаками **#,** например:

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

### 12.4 Переменные

**Переменная** – это удобное хранилище, указывающее на место в памяти компьютера, где вы можете хранить информацию, которая может измениться в процессе выполнения скрипта. Неважно где именно в памяти находится переменная, вы просто должны обращаться к ней по имени, чтобы прочитать или изменить ее значение.

В **VBScript**, переменные всегда представлены типом данных ***Variant***.

#### 12.4.1 Объявление переменных.

Вы можете объявлять переменные в скрипте явно, используя, оператор **Dim**, **Public** или **Private,** например: **Dim** DegreesFahrenheit

Можно объявить сразу несколько переменных: **Dim** Top, Bottom, Left, Right

Переменные можно объявлять неявно, используя их имя в скрипте. Это не очень хороший способ, так как можно допустить синтаксические ошибки в имени переменной где-либо в скрипте, что может повлечь за собой ошибки в работе скрипта. Чтобы исключить такую возможность используйте конструкцию **Option Explicit**. Эта конструкция запрещает использование необъявленных явно переменных. **Option Explicit** должна быть первой строкой скрипта. При использовании этой конструкции вы должны явно объявлять все переменные.

Например: **Option Explicit** ' Обязательно явное объявление переменных.
**Dim MyVar** ' Объявляем переменную.
**MyInt = 10** ' Ошибка – переменная не объявлена явно.
**MyVar = 10** ' Все правильно – переменная объявлена.

### 12.4.2 Правила наименования переменных

**Имя переменной** должно удовлетворять следующим правилам:

- Начинаться с символа латинского алфавита
- Не содержать внутри себя знаков препинания
- Не превышать (по длине) 255 символов
- Быть уникальным в области видимости, в которой объявлена

### 12.4.3 Область видимости и время жизни переменных

**Область видимости** переменной определяется местом, где она объявлена. Когда вы объявляете переменную в процедуре, то только код внутри этой процедуры может «видеть» и изменять значение этой переменной. Это называется локальной областью видимости, а такая переменная называется **локальной** или переменной уровня процедуры. Если же вы объявляете переменную вне процедуры, переменная становится доступна в любом месте скрипта, включая процедуры, функции и т.д. Такие переменные называются **глобальными**, либо переменными области видимости скрипта.

Время жизни переменной определяется областью видимости. Время жизни глобальной переменной исчисляется с момента их объявления до завершения работы скрипта. Переменная уровня процедуры существует с момента объявления и до выхода из процедуры. При выходе из процедуры переменная уничтожается. Локальные переменные идеально подходят для временного хранения данных. У вас может несколько локальных переменных с одинаковыми именами в различных процедурах, так как они распознаются только в той процедуре, в которой объявлены.

### 12.4.4 Присвоение значений переменным

Для присваивания значения переменной используется оператор присваивания(**=**) вида:

**<имя переменной> = <выражение>**

Например:

**B = 200**

В примере в качестве выражения используется константа 200.

### 12.4.5 Скалярные переменные и массивы

Переменная, содержащая одиночное значение называется **скалярной**.

Для решения различных задач часто необходимо некоторое множество однотипных значений с одним наименованием. Совокупность одноименных значений называется **массивов**.

Объявление массивов и скалярных переменных одинаково, за исключением того, что при объявлении массивов после имени располагаются скобки, в которых указывается размер массива. Размер массива указывает количество элементов массива и равен указанному в скобках значению плюс 1, так как **все массивы в VBScript индексируются с нуля**. Например, объявление одномерного массива A, содержащего 11 элементов: **Dim A(10)**

Вы можете присваивать данные каждому элементу массива, используя индекс в имени массива: **A(0) = 256**

**A(1) = 324**

**A(2) = 100**

**...**

**A(10) = 55**

Таким же образом вы можете получить данные из массива, например: **SomeVariable = A(8)**

Приведенный в примерах тип массива называется одномерным массивом фиксированного размера.

В следующем примере показан двух размерный массив из 6 строк и 11 столбцов:

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

В двух размерных массивах первый параметр всегда количество строк, второй – количество столбцов.

**ВНИМАНИЕ!!!**

**Размер массива не ограничен.**
**Размерность массива ограничена – максимум 60.**

Для объявления **динамических массивов**, размер которых может измениться в процессе выполнения скрипта, в конструкции **Dim** в скобках не указывается размер и размерность массива. Например:

```
Dim MyArray()
```

Чтобы использовать динамический массив в коде скрипта конструкцией **ReDim** должны быть определены его размерность и количество элементов. В следующем примере устанавливается первоначальный размер массива равный 25, затем он изменяется до 30.

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

Ключевое слово **Preserve** указывает, что запрещено изменять размерность массива.

**ВНИМАНИЕ!!!**

**Можно многократно изменять размер массива, но следует помнить, что если вы делаете массив меньше, то вы теряете данные в удаленных элементах массива – они не восстановятся, если размер массива увеличить вновь.**

## 12.5 Выражения и операции

**Выражение** – это последовательность констант и переменных (а также и функций), соединенных знаками операций. Например: **a * b -c**

Выражения используются в операторах присваивания, условных и циклических операторах, как аргументы вызова процедур и функций. Например: **Result = a * b -c**

Для записи выражений **VBScript** имеет полный набор операций, включая арифметические и логические операции, операции сравнения и объединения.

### 12.5.1 Приоритет операций

Арифметические и логические операции выполняются в порядке, указанном в таблицах 12.5.1, 12.5.2 (приоритет понижается сверху вниз).

Таблица 12.5.1. Арифметические операции

| Описание | Символ |
|---|---|
| Возведение в степень: a^b 'a в степени b | ^ |
| Унарный минус: -b | - |
| Умножение: a*b | * |
| Деление: a/b | / |
| Целочисленное деление: a\b 'результат – целое от деления a на b | \ |
| Остаток от деления: a Mod b | Mod |
| Сложение: a + b | + |
| Вычитание: a - b | - |

Таблица 12.5.2 Логические операции

| Описание | Символ |
|---|---|
| Логическое отрицание | Not |
| Логическое И | And |
| Логическое ИЛИ | Or |
| Логическое ИСКЛЮЧЕНИЕ | Xor |
| Логическая эквивалентность | Eqv |
| Логическая импликация | Imp |

Все операции сравнения имеют одинаковый приоритет.

Таблица 12.5.3 Операции сравнения

| Описание | Символ |
|---|---|
| Равно | = |
| Не равно | <> |
| Меньше чем | < |
| Больше чем | > |
| Меньше или равно | <= |
| Больше или равно | >= |
| Эквивалентность объектов | Is |

Оператор **Is** сравнивает ссылки на объекты, но не сравнивает значения этих объектов.

Операция **объединения строк** – **&**. Например: **"Строка1" & "Строка2"**
Приоритет операции объединения ниже всех арифметических операций, но выше чем у операций сравнения.

## 12.6 Операторы управления. Условные операторы

В **VBScript** существует два типа условных операторов:

- **If … Then … Else**
- **Select Case**

### 12.6.1 Оператор If … Then … Else

Оператор **If … Then … Else** используется для разветвления вычислительного процесса в зависимости от значения условия (логического выражения).

Синтаксис:

**If < логическое выражение > Then < оператор1 > Else < оператор2 >**

Если логическое выражение истинно (**True**), то выполняется оператор1, иначе (значение выражения ложно – **False**) – оператор2.

В случае, когда по условию необходимо выполнить только один оператор используется однострочный синтаксис. Например:

```
Dim myDate
myDate = #2/13/95#
If myDate < Now Then myDate = Now
```

Для того чтобы выполнить по условию более одного оператора, необходимо использовать многострочный синтаксис, который включает конструкцию **End If**. Например:

```
If value = 0 Then
  AlertLabel.ForeColor = vbRed
  AlertLabel.Font.Bold = True
  AlertLabel.Font.Italic = True
End If
```

Оператор **If** может включать вложенные операторы **If**. В этом случае вместо ключевого слова **Else** необходимо использовать ключевое слово **ElseIf**. Например:

```
If value = 0 Then                          ' ЕСЛИ value = 0 ТО
  MsgBox value                             ' Выполняем MsgBox value
ElseIf value = 1 Then                      ' ИНАЧЕ ЕСЛИ value = 1 ТО
      MsgBox value                         ' Выполняем MsgBox value
    ElseIf value = 2 then                  ' ИНАЧЕ ЕСЛИ value = 2 ТО
          Msgbox value                      ' Выполняем инструкцию MsgBox value
        Else                               ' ИНАЧЕ
          Msgbox "Value out of range!"     ' Выполняем MsgBox "Value out of range!"
End If                                     ' Конец оператора If
```

Можно добавлять ключевое слово **ElseIf** столько раз, сколько это необходимо. Однако при наличии большого числа условий оператор **If** может стать слишком громоздким. Лучшим решением для выбора из нескольких альтернатив является оператор **Select Case**.

### 12.6.2 Использование Select Case

Оператор **Select Case** используется для разветвления вычислительного процесса по нескольким направлениям в зависимости от значения выражения. **Select Case** обеспечивает схожую с **If … Then … ElseIf** функциональность, но делает код скрипта более эффективным и читабельным.

Синтаксис:

```
Select Case <выражение>
  Case <значение1>
    <операторы1>
  Case <значение2>
    <операторы2>
     . . .
  Case <значениеN>
    <операторыN>
   Case Else            ‘Иначе – ни одно Case-значение не сравнилось
    <операторы>         ‘ со значением выражения
End Select
```

Например:

```
Select Case Text                'Вычисляется значение Text
  Case "MasterCard"             'Если Text = "MasterCard"
    DisplayMCLogo               'Выполняются DisplayMCLogo и
    ValidateMCAccount           'ValidateMCAccount
  Case "Visa"                   'Если Text = "Visa"
    DisplayVisaLogo             'Выполняются DisplayVisaLogo и
    ValidateVisaAccount         'ValidateVisaAccount
  Case "American Express"       'Если Text = " American Express"
    DisplayAMEXCOLogo           'Выполняются DisplayAMEXCOLogo и
    ValidateAMEXCOAccount       'ValidateAMEXCOAccount
  Case Else                     'Иначе выполняются
    DisplayUnknownImage         'DisplayUnknownImage и
    PromptAgain                 'PromptAgain
End Select                      'Завершение оператора Select Case
```

## 12.7 Операторы управления. Циклы

Операторы циклов организуют многократное повторение группы операторов.
В **VBScript** доступны следующие разновидности операторов циклов:

- **Do...Loop**
- **While...Wend**
- **For...Next**
- **For Each...Next**

### 12.7.1 Оператор Do...Loop

Операторы, входящие в цикл выполняются пока условие истинно (цикл-ПОКА – **Do While… Loop**) или до тех пор, пока условие не станет истинным (цикл-ДО – **Do Until… Loop**).

**Цикл-ПОКА**.

Синтаксис:

```
Do While <логическое выражение>   'ПОКА выражение истинно,
  <операторы>                     'операторы выполняются
Loop
```

или

```
Do
  <операторы>                     'Операторы выполняются
Loop While <логическое выражение> 'ПОКА выражение истинно
```

В первом случае условие проверяется до того, как цикл начал выполняться, и операторы могут не выполниться ни разу. Во втором случае гарантировано выполнение операторов хотя бы один раз. Например:

```
Dim counter, myNum
counter = 0
myNum = 20
Do While myNum > 10
  myNum = myNum - 1
  counter = counter + 1
Loop
MsgBox "The loop made " & counter & " repetitions."
```

или

```
Dim counter, myNum
counter = 0
myNum = 9
Do
  myNum = myNum - 1
  counter = counter + 1
Loop While myNum > 10
MsgBox "The loop made " & counter & " repetitions."
```

**Цикл-ДО**

Синтаксис:

```
Do Until <логическое выражение>      'ДО тех пор, пока выражение не станет истинно,
  <операторы>                        'операторы выполняются
Loop
```

или

```
Do
  <операторы>                        'Операторы выполняются
Loop Until <логическое выражение>    ' ДО тех пор, пока выражение не станет истинно
```

Например:

```
Dim counter, myNum
counter = 0
myNum = 20
Do Until myNum = 10
    myNum = myNum - 1
    counter = counter + 1
Loop
MsgBox "The loop made " & counter & " repetitions."
```

или

```
Dim counter, myNum
counter = 0
myNum = 1
Do
  myNum = myNum + 1
  counter = counter + 1
Loop Until myNum = 10
MsgBox "The loop made " & counter & " repetitions."
```

### 12.7.2 Прерывание цикла **Do...Loop**

При построении алгоритмов возникает ситуация, когда надо прервать выполнение цикла и выйти из него. Обычно для этого используют оператор **If...Then...Else** (проверка условия выхода из цикла) и оператор **Exit Do** (выйти из цикла).

Синтаксис (на примере цикла-ДО):

```
Do Until <логическое выражение>
  <операторы>
  If <условие выхода из цикла>Then Exit Do
  <операторы>
Loop
```

Оператор **Exit Do** передает управление на оператор, стоящий в программе после **Loop**.

В следующем примере переменной **myNum** присваивается значение, создающее бесконечный цикл. Конструкция **If...Then...Else** проверяет условие и вызывает выполнение оператора **Exit Do,** тем самым прерывая цикл и предотвращая зацикливание программы.

```
Dim counter, myNum
counter = 0
myNum = 9
Do Until myNum = 10
  myNum = myNum - 1
  counter = counter + 1
  If myNum < 10 Then Exit Do
Loop
MsgBox "The loop made " & counter & " repetitions."
```

### 12.7.3 Оператор **While … Wend**

Цикл **While … Wend** полностью идентичен по функциональности циклу **Do … Loop**.

### 12.7.4 Оператор **For … Next**

Цикл **For … Next** позволяет выполнять блок инструкций определенное число раз. Для этого используется переменная-счетчик, которая уменьшает или увеличивает свое значение на каждой итерации.

Синтаксис:

```
For <переменная-счетчик> = <начальное значение> TO <конечное значение>
  <операторы>  'операторы выполняются при всех значениях переменной-счетчик,
               'которая увеличивается на 1 от начального значения до конечного
Next
```

или

```
For <переменная-счетчик> = <начальное значение> TO <конечное значение>
                                Step <выражение>
  <операторы>  'операторы выполняются при всех значениях переменной-счетчик,
               'которая изменяется от начального значения до конечного на величину,
               'равную вычисленному значению выражения
Next
```

Следующие примеры показывают использование оператора **For**.

```
Dim j, total
total = 0
For j = 1 To 5          'На каждом шаге цикла значение j увеличивается на 1
  total = total + j     'На каждом шаге цикла total увеличивается на значение j
Next                    'Результат цикла – total = 0+1+2+3+4+5 = 15
```

```
Dim j, total
total = 0
For j = 1 To 5 Step 2     'На каждом шаге цикла значение j увеличивается на 2
  total = total + j       'На каждом шаге цикла total увеличивается на значение j
Next                      'Результат цикла – total = 0+1+3+5 = 9


Dim j, total
total = 0
For j = 5 To 1 Step -2    'На каждом шаге цикла значение j уменьшается на 2
  total = total + j       'На каждом шаге цикла total увеличивается на значение j
Next                      'Результат цикла – total = 0+5+3+1 = 9
```

Выйти из цикла **For ... Next** до его завершения можно, используя оператор **Exit For**.

### 12.7.5 Оператор For Each...Next

Цикл **For Each...Next** выполняет заданные операторы для каждого элемента коллекции объектов или элемента массива. Это особенно полезно, если заранее неизвестно, сколько элементов в коллекции.

Синтаксис:

```
For Each <переменная> in <коллекция>
  <операторы>  'операторы выполняются для каждого элемента коллекции
Next
```

Например:

```
Dim d                                       'Объявление переменной – коллекции
Set d = CreateObject("Scripting.Dictionary") 'Создание пустой коллекции
d.Add "0", "Athens"                         'Добавление в коллекцию
d.Add "1", "Belgrade"'                      'ключей и
d.Add "2", "Cairo"                          'элементов

For Each j in d                             'Перебор всех элементов коллекции
  Document.frmForm.Elements(j).Value = d.Item(j)
Next
```

## 12.8 Подпрограммы

Подпрограммы – это логически законченная последовательность действий по обработке данных. Подпрограммы позволяют избежать дублирования кода и уменьшают сложность программ.

В VBScript существует два вида подпрограмм: **Sub** и **Function**.

Синтаксис:

```
Sub <наименование подпрограммы>( ) 'Заголовок подпрограммы
или
Function Sub <наименование подпрограммы> (<параметры>)
  <операторы>        'Операторы подпрограммы
End Sub              'Завершение текста подпрограммы

Function <наименование подпрограммы>( ) 'Заголовок подпрограммы
или
Function <наименование подпрограммы> (<параметры>)
  <операторы>        'Операторы подпрограммы
End Function         'Завершение текста подпрограммы
```

Параметры используются для передачи данных в подпрограмму.

### 12.8.1 Подпрограмма Sub

Подпрограмма **Sub** – это процедура, которая не возвращает значений в вызывающую программу (подпрограмму).

Например:

```
Sub ConvertTemp()
  Dim temp, cels
  temp = InputBox("Please enter the temperature in degrees F.", 1)
  cels  = (temp – 32) ∗ 5 / 9
  MsgBox "The temperature is " & cels & " degrees C."
End Sub
```

Вызов **Sub** осуществляется отдельным оператором вызова, в котором используется наименование процедуры.

Например, оператор вызова процедуры **MyProc** с параметрами **first_arg, second_arg** выглядит так:

```
MyProc first_arg, second_arg
```

При вызове процедуры с помощью оператора **Call** передаваемые параметры необходимо заключить в скобки. Например:

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

### 12.8.2 Подпрограмма Function

Подпрограмма **Function** – это функция, которая может возвращать значение.

Возвращаемое значение всегда имеет тип **Variant**. Для возвращения значения в теле функции должен быть хотя бы один оператор присваивания вида: **<имя функции> = <выражение>**. Возвращаемое значение равно результату вычисления выражения.

Например:

```
Function Celsius(fDegrees)          'Параметр fDegrees – входные данные функции
  Celsius = (fDegrees – 32) ∗ 5 / 9 'Возвращаемое значение функции
End Function
```

Вызов **Function** осуществляется по наименованию функции из других операторов.

Например:

вызов функции **Celsius** с параметром **fDegrees** в операторе присваивания:

```
Temp = Celsius(fDegrees)
```

вызов функции **Celsius** с параметром **fDegrees** в операторе вывода – системной процедуре **MsgBox**:

```
MsgBox "The Celsius temperature is " & Celsius(fDegrees) & " degrees."
```

## 12.9 Переменные для связи с БД

Наряду с переменными VBScript, которые обсуждались выше, в скрипты системы КРУГ-2000 включен специальный вид переменных – **переменные для связи с базой данных** или сокращенно **БД-переменные**.

Эти переменные **всегда являются глобальными** для скрипта независимо от того, где они объявлены, и используют свой собственный синтаксис и свои правила именования, отличные от тех, что приняты в VBScript.

БД-переменные могут быть либо **ссылкой на переменную** в базе данных, либо **значением** типа ***Variant***. На то является ли переменная ссылкой или же задается значением, указывает префикс (первый символ) в имени переменной. Это могут быть либо символы @ и #, если БД-переменная – ссылка, либо **%,** если БД-переменная – значение.

Если переменная с префиксом @ имеет ссылку на атрибут переменной БД, тогда изменение ее значения будет сопровождаться сообщением в роллинг. Для того чтобы сообщения в роллинг не отсылались, переменная должна иметь префикс #.

Например:

```
Sub DoSomething()
  ' Присваиваем переменной в базе данных, на которую
  ' ссылается БД-переменная @Температура_воды, значение 30
  @Температура_воды = 30
  ' Присваиваем БД-переменной значение 0.5
  %Коэффициент_001 = 0.5
End Sub
```

Имя переменной, следующее за префиксом, может включать в себя символы русского и латинского алфавита, цифры и знаки подчеркивания. Имя БД-переменной, включая префикс, не может превышать 30 символов.

В случае, когда имя переменной превышает 30 символов, символы, следующие за 30-ым, отсекаются транслятором.

Например,

использование следующих имен БД-переменных:

**@Ссылка_на_переменную_номер_012 = 10**

**@Ссылка_на_переменную_номер_013 = 20**

будет распознано транслятором как одна и та же переменная

**@Ссылка_на_переменную_номер_01 = 10**

**@Ссылка_на_переменную_номер_01 = 20**

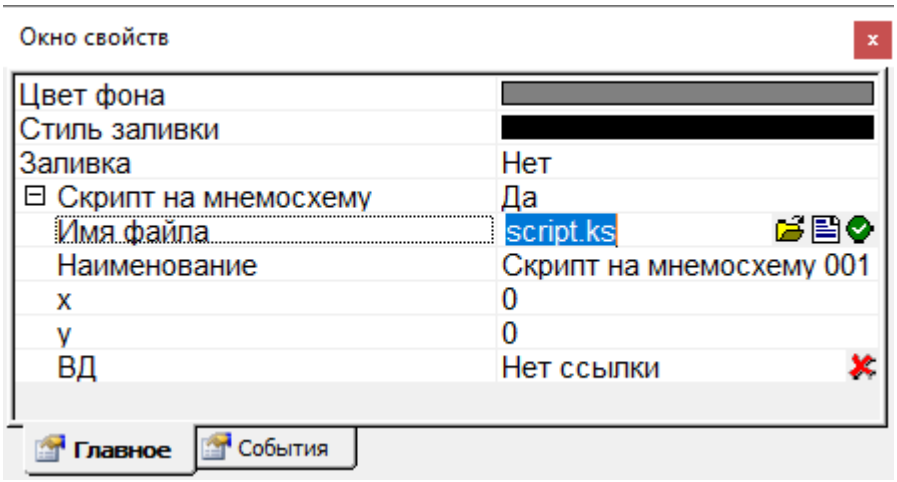
**Редактор скриптов** отслеживает это и «подсказывает» пользователю, что в имени переменной есть ошибка.

```
'Всё правильно
 @Ссылка_на_переменную_номер_1 = 10

'Ошибка имя переменной больше 30 символов
 @Ссылка_на_переменную_номер_012 = 10
```

Привязка БД-переменных к базе данных осуществляется в **Окне свойств** проекта, где отображаются все БД-переменные, которые используются в скрипте.

БД-переменным, которые задаются значением в **Окне свойств** можно присвоить начальные значения: X, Y – задаются значением, ВД – задается ссылкой




12.9.0 Объявление БД-переменных

БД-переменные, как и переменные VBScript, в общем случае не требуют предварительного объявления. Но, тем не менее, эта возможность предусмотрена. Для объявления БД-переменных используется конструкция **<<...>>** (два открывающих знака «меньше» и два закрывающих знака «больше»), которая называется **секция объявления БД-переменных**.

Например:

**<<%Давление, @Температура>>**

Такие секции могут находиться в любом месте скрипта и в любом количестве. Необходимо просто перечислить в них имена переменных вместе с префиксами. Перечислять можно, через любые символы, которые не могут содержаться в имени переменной. Например:

запятая, точка, пробел и т.д. Также стоит отметить, что использование конструкции **Option Explicit** из языка VBScript не делает обязательным предварительное объявление БД-переменных.

### 12.9.1 Использование БД-переменных

Над БД-переменными можно производить все операции, что и над обычными переменными VBScript. Например, присваивать им значения; использовать в арифметических операциях; передавать значения переменных в качестве параметров при вызове процедур и функций.

Например:

```
Sub OnTimer ()
  If @Турбина_смещение > 145 And Not bDown Then
    @Турбина_смещение = @Турбина_смещение - 1
    @Турбина_смещение2 = @Турбина_смещение2 + 1
     %Угол_1 = %Угол_1 + 0.01
     %Угол_2 = %Угол_2 + 0.01
  EndIf
End Sub
```

### 12.9.2 Переменные, задаваемые значением

БД-переменные, которые задаются значением, предназначены для хранения информации, которая может потребоваться после завершения скрипта, или использования в качестве входных параметров скрипта (как это можно сделать описывается далее в разделе «Функция Преобразование скриптом»).

### 12.9.3 Переменные, задаваемые ссылкой на БД

В случае если БД-переменная является ссылкой на базу данных, то значение, присваиваемое ей, записывается в тот элемент базы данных, на который ссылается эта переменная.

Например, БД-переменная **@Текст** ссылается на атрибут **Текст** графического примитива.

После выполнения скрипта со строкой:

```
...
@Текст = "Новый текст"
...
```

В атрибут **Текст** запишется значение **"Новый текст"**.

При присваивании значения БД-переменной, которая задается ссылкой, какой-либо другой переменной, либо при передаче ее в качестве параметра функции или процедуре, происходит чтение значения элемента базы данных, на который эта переменная ссылается.

Если ссылка на БД не задана или ошибочна, информация об этом отображается в **Редакторе скриптов** на закладке **Информация времени выполнения**.

## 12.10 Виды скриптов

В системе КРУГ-2000 скрипты разделяются на три категории, которые различаются между собой по функциональному назначению.

- Скрипт на мнемосхему
- Функция «Преобразование скриптом»
- Реакция «Выполнить скрипт»

**ВНИМАНИЕ!!!**

**Для всех категорий действует общее правило – скрипт выполняется только тогда, когда открыта мнемосхема, к которой он «привязан».**

**На Станции оператора (Графический интерфейс) скрипт начинает свою работу сразу после открытия мнемосхемы и завершается, когда мнемосхема закрывается.**

**В Генераторе динамики скрипт начинает свою работу при запуске мнемосхемы на имитацию и завершается при возвращении в режим редактирования. При открытии мнемосхемы скрипт лишь предварительно транслируется (подключаются библиотеки, происходит поиск БД-переменных и другие действия).**

### 12.10.1 Общие принципы работы скриптов в системе КРУГ-2000

Все скрипты, как уже говорилось выше, «живут» пока открыта мнемосхема, к которой они «привязаны». Что означает «привязаны»? Функция **Преобразование скриптом** и реакция **Выполнить скрипт** могут быть назначены только на какой-либо элемент динамики, который находится на мнемосхеме. Каждый элемент динамики принадлежит мнемосхеме, на которой он расположен. Соответственно при закрытии мнемосхемы, либо при удалении динамического элемента, закрывается и назначенный на этот динамический элемент скрипт.

Аналогичная ситуация происходит и со **Скриптом на мнемосхему**. Он назначается непосредственно на мнемосхему и создается/уничтожается при ее открытии/закрытии.

При открытии мнемосхемы, а в Генераторе динамики при запуске на имитацию, все скрипты, принадлежащие этой мнемосхеме, независимо от вида, выполняют следующую последовательность действий:

1. Предварительный разбор скрипта. Поиск переменных, подключение библиотек

   (этап может быть пропущен, если скрипт не изменился с момента последней загрузки)

2. Инициализация. Выполняется та часть скрипта, которая расположена в области видимости скрипта (смотрите подраздел 12.4.3), другими словами выполняется все, что не является частью процедур, функций, классов и т.п.

   В следующем примере жирным шрифтом отмечены места, которые будут выполнены при инициализации, а в комментариях в скобках цифры указывают на порядок выполнения:

```
<<@ВД, %x, %y>> 'Вызывается при предварительном разборе скрипта
'Инициализация переменных, описание функций и процедур
%x = 0:%y = 0                '(1)
Dim somevariable             '(2)
somevariable = 0             '(3)

Sub SomeSub_001 ()
    %x = 10                  '(5) Выполняется после вызова (4)
End Sub
```

```
Sub SomeSub_002 ()
  %y = 20 'Не выполняется, т.к. находится в области видимости процедуры
End Sub

' Выполняется так, как находится в области видимости скрипта
SomeSub_001              '(4) Вызывает процедуру SomeSub_001 (5)

'Секция выполнения (используется в реакциях и функциях преобразования)
{{
  @ВД = %x*%y   'Не выполняется – область видимости секции выполнения
  SomeSub_002   'Не выполняется – область видимости секции выполнения
}}

Trace "Инициализация скрипта завершена"   '(6)

Sub OnFinishScript ()
  Trace "Завершение скрипта"              'Не выполняется
End Sub
```

**ВНИМАНИЕ!!!**

**В функции «Преобразования скриптом» сразу после инициализации скрипта вызывается код из *секции выполнения*, а затем осуществляется переход в режим ожидания событий.**

**После инициализации скрипт не выполняет никаких действий, пока не произойдет какое-либо событие, сопоставленное с ним.**

**Так например, в функции реакции «Выполнить скрипт» скрипту может быть сопоставлено какое-либо событие из доступных событий функции реакции: нажатие клавиши мыши, попадание курсора и другие. При возникновении такого события скрипт, назначенный на функцию реакции, выполняет код расположенный в *секции выполнения*.**

При закрытии мнемосхемы, перед завершением скрипта вызывается код из процедуры **OnFinishScript**. Эта процедура является обычной Sub-процедурой языка VBScript. **OnFinishScript** *не должна* иметь параметров и *должна* быть объявлена в скрипте следующим образом:

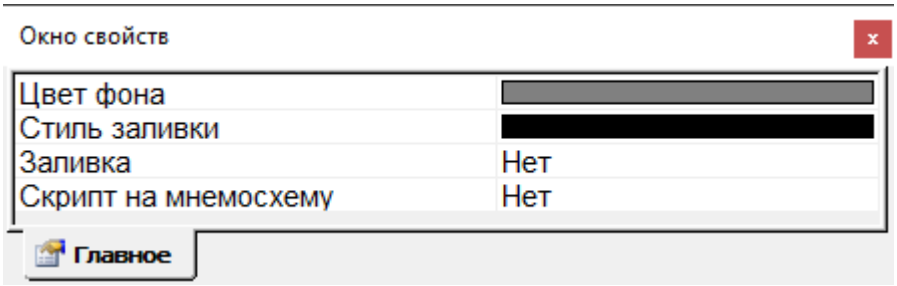
```
Sub OnFinishScript ()
  <операторы>          'Выполнение завершающих действий
End Sub
```

Процедура **OnFinishScript** в скрипте может отсутствовать.

### 12.10.2 Скрипт на мнемосхему

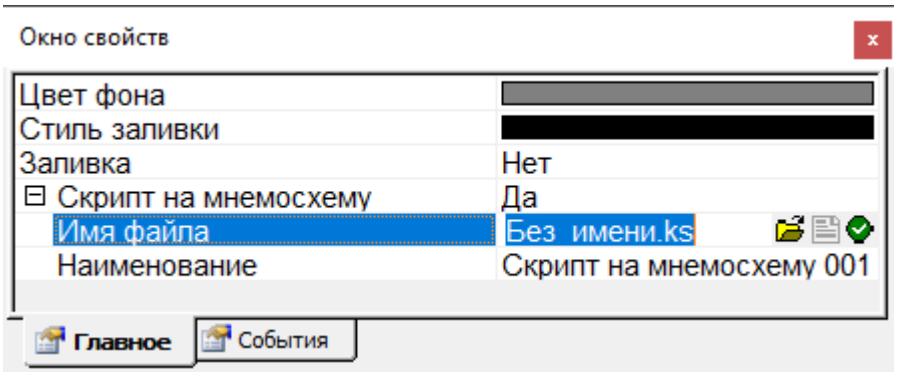
На каждую мнемосхему проекта (и на любой другой тип схемы: рабочий стол, шаблон прибора и т.д.) можно назначить скрипт, который будет глобальным для данной мнемосхемы. Этот скрипт называется **Скриптом на мнемосхему**.

Каждой мнемосхеме может быть назначен только *один* скрипт этого вида. Это можно сделать в **Окне свойств**. Для этого необходимо открыть мнемосхему в **Генераторе динамики**, щелкнуть левой клавишей мыши в любом свободном месте на мнемосхеме (не на элементе). В **Окне свойств** появится следующая информация.



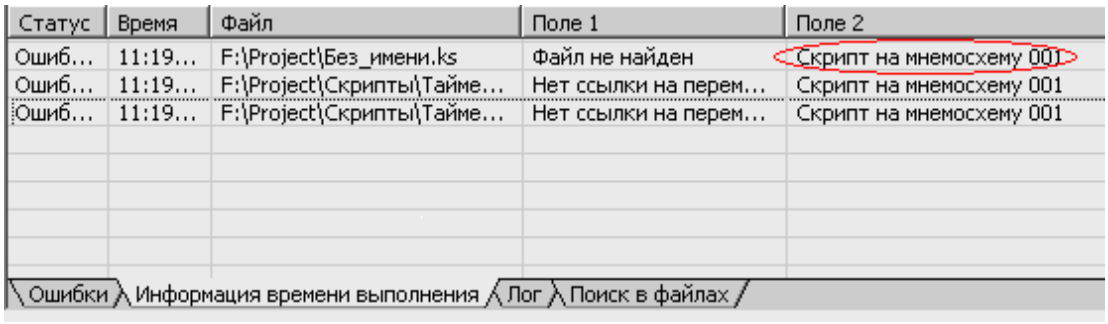

В свойстве **Скрипт на мнемосхему** следует указать **Да.** В результате в **Окно свойств** мнемосхемы добавится закладка **События**.

Для того чтобы указать файл скрипта, следует раскрыть значок ⊞ в свойстве **Скрипт на мнемосхему.**




Имя файла можно набрать вручную в поле **Имя файла**, либо воспользовавшись стандартным диалогом открытия файла нажав . Кроме кнопки  в поле **Имя файла** присутствует также кнопка  – открытие файла скрипта с указанным именем в **Редакторе скриптов**. Если файл с таким именем на диске не найден, эта кнопка не активна. Кнопка  предназначена для обновления скрипта (выполняется предварительная трансляция скрипта).

Поле **Наименование** по умолчанию содержит имя скрипта, уникальное для данной мнемосхемы. Его можно изменять, однако, желательно не задавать скриптам одинаковые имена, чтобы была возможность «распознать» скрипт на закладке **Информация времени выполнения** в Редакторе скриптов.

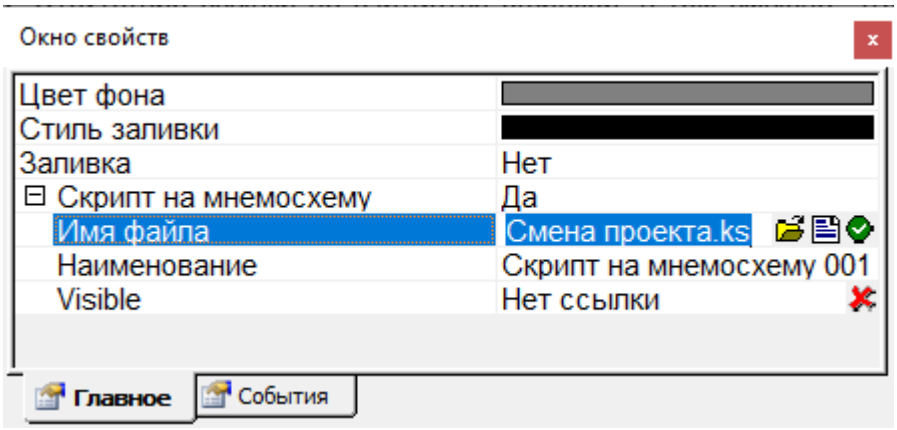



Имя файла скрипта может быть абсолютным (т.е. полный путь + имя файла, например**: F:\Project\MyScript.ks)** или относительным.

Для относительного имени файла базовой считается папка, в которой расположен файл проекта (***.kgp**). Например, пусть файл проекта находится в папке **E:\MyProject**, тогда, если в Окне свойств в поле **Имя файла** указать **Scripts\MyScript.ks**, то для данного проекта имя файла скрипта будет расшифровано как **E:\MyProject\Scripts\MyScript.ks**. Использование относительных имен для файлов скриптов позволяет не привязывать скрипты проекта к конкретному расположению на внешнем носителе.

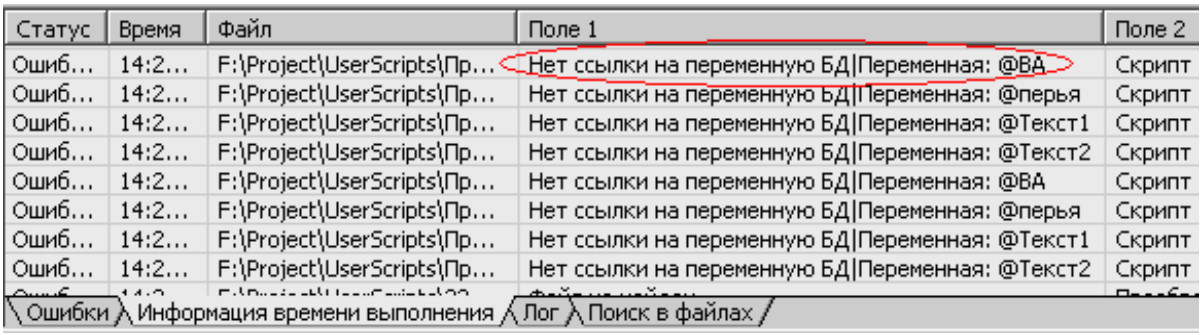
В случае если имя файла начинается с **%PathToLib%** (переменная окружения, указывает на путь к библиотеке), строка **%PathToLib%** заменяется на путь к библиотеке системы КРУГ-2000. Например: если **%PathToLib% – «F:\Program Files\Krug2000\Library»,** то относительный путь **%PathToLib%\Script\MyScript.ks** будет расшифрован как **F:\Program Files\Krug2000\Library\Script\MyScript.ks**

После того как файл скрипта задан, если в коде скрипта присутствуют БД-переменные, то они без префиксов @ и **%** отображаются в Окне свойств.




Переменным, задаваемым ссылкой на БД, необходимо назначить эту ссылку через диалог **Выбор ссылки**. Отсутствие ссылки не считается ошибкой, в том смысле, что выполнение скрипта будет продолжено, но все операции (сложения, умножения и другие) с такой переменной не приведут к ожидаемому результату.

В Редакторе скриптов на закладке **Информация времени выполнения** отображается информация об ошибочных или не заданных ссылках.




Для того чтобы, скрипт на мнемосхему начал что-либо делать, необходимо назначить ему хотя бы одно обрабатываемое событие. Если этого не сделать то, после запуска скрипта будет выполнена только его инициализация.

Скрипту на мнемосхему может быть назначено несколько видов событий. В настоящее время поддерживается три:

- **Таймер**
- **Уведомление**
- **Сообщение.**

### 12.10.3 События скрипта на мнемосхему

Назначение событий происходит на закладке **События** в Окне свойств мнемосхемы, с помощью кнопок панели инструментов(слева направо): назначить обработку события ***таймер***, назначить обработку события ***уведомление***, назначить обработку события ***сообщение***, удалить отмеченное событие.


Все события должны быть сопоставлены со своим ***обработчиком***.

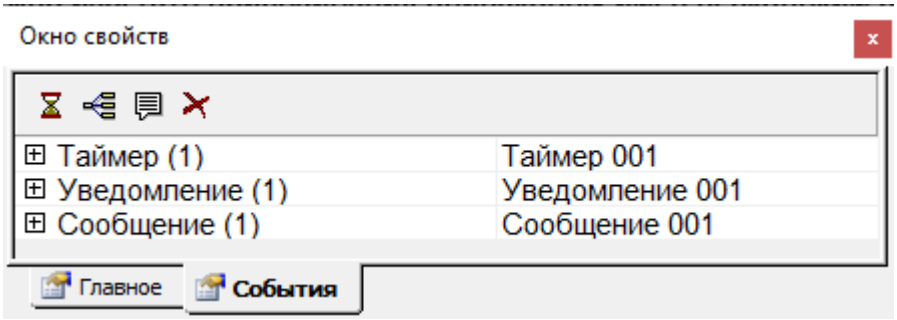
Что такое обработчик? Это обыкновенная Sub-процедура из языка VBScript, которая вызывается каждый раз при возникновении какого-либо события. Процедура располагается в глобальном пространстве имен и не имеет параметров:

```
Sub OnSomeEvent ()
  <операторы>          'Здесь размещается код скрипта для обработки события
End Sub
```

Следует отметить, что назначение обработчика должно выполняться для всех видов событий.

Назначение событиям соответствующих обработчиков рассматривается ниже, на примере события **Таймер**.

Другой общей характеристикой события является его **имя**. По умолчанию оно уникально в пределах мнемосхемы. Имя события можно изменять.




**ВНИМАНИЕ!!!**

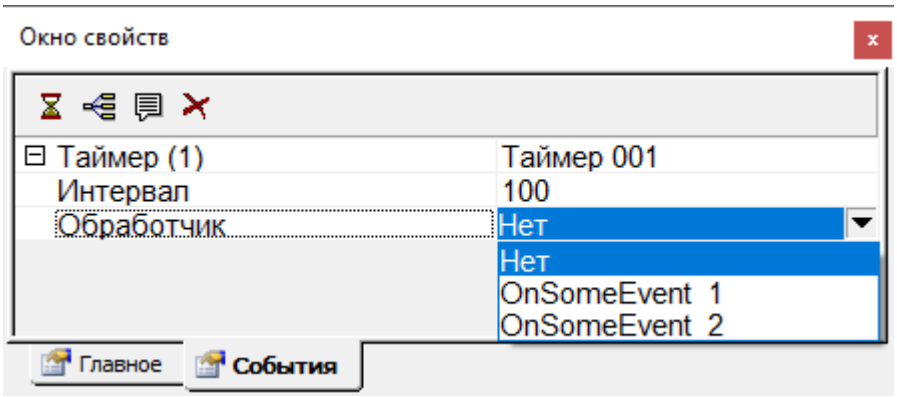
**Следует помнить, что имя события используется при выводе информации времени выполнения в Редактор скриптов и в некоторых *встроенных функциях.* Поэтому нужно избегать присваивания одинаковых имен нескольким событиям.**

12.10.3.1 Событие «Таймер»

Иногда бывает необходимо выполнять какие-либо действия через определенный интервал времени. Событие **Таймер** предоставляет такую возможность.

Принцип работы события прост: через заданный интервал времени выполняется код из сопоставленного данному событию обработчика. Обработчик выполняется не единожды, а каждый раз по истечении интервала. Интервал задается в миллисекундах.

Выбор обработчика осуществляется в Окне свойств, в поле «Обработчик» (рисунок 12.10.3.3), в поле «Интервал» необходимо задать время в миллисекундах (по умолчанию 250 мс), через которое будет выполняться код процедуры-обработчика.




Список доступных обработчиков определяется на этапе предварительной трансляции скрипта, который происходит при открытии мнемосхемы, изменении имени файла скрипта в поле **Имя файла** или при нажатии кнопки **Обновить** .

Например, пусть на мнемосхему назначен следующий скрипт:

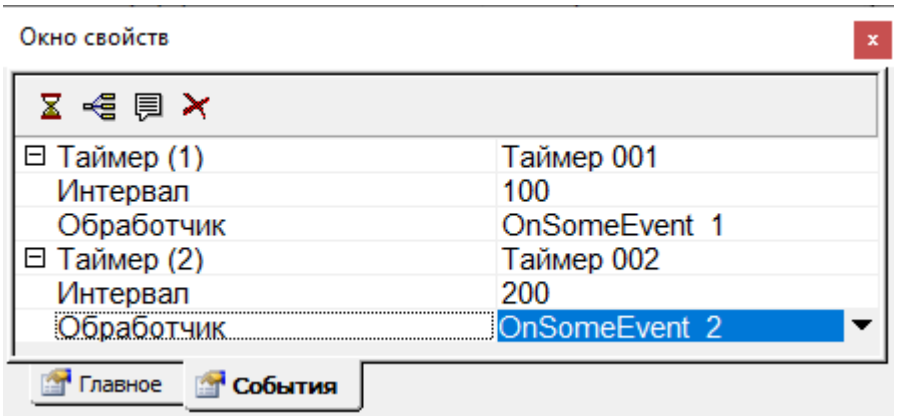
```
<<%x, %y>>
'Инициализация переменных, описание функций и процедур
%x = 0 : %y = 0
Dim somevariable
somevariable = 0
Sub OnSomeEvent_1 ()
```

```
    %x = somevariable + 10
End Sub

Sub OnSomeEvent_2 ()
    %y = somevariable - 20
End Sub
```

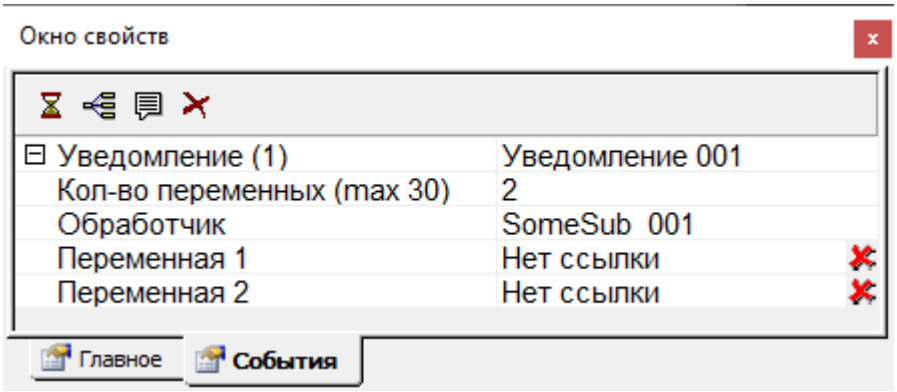
В качестве обработчиков событий в этом скрипте могут быть использованы процедуры: **OnSomeEvent_1** и **OnSomeEvent_2**, так как они расположены в глобальном пространстве имен и у них нет параметров.

Если для данного скрипта задать два события **Таймер** с интервалами 100 и 200 мс, а в качестве обработчиков для первого события указать **OnSomeEvent_1**, а для второго – **OnSomeEvent_2**, то получим, что код **%x = somevariable + 10** будет выполняться каждые 100 мс, а код **%y = somevariable - 20** – каждые 200 мс.

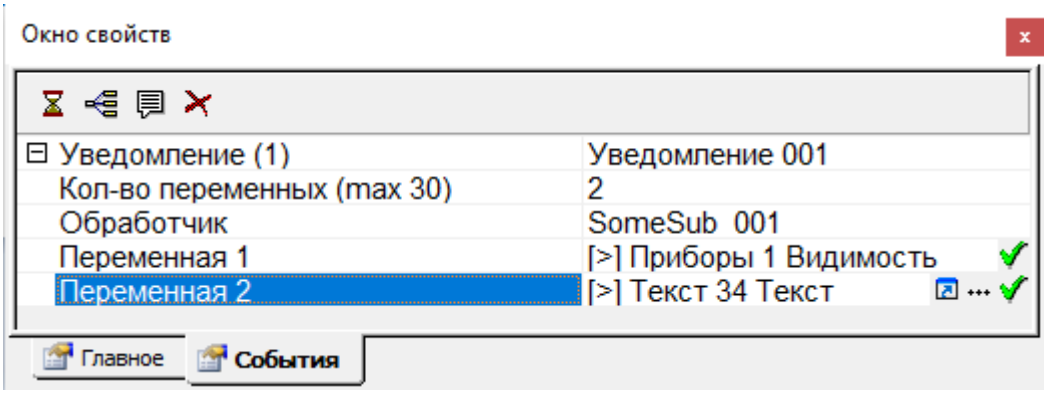



### 12.10.3.2 Событие «Уведомление»

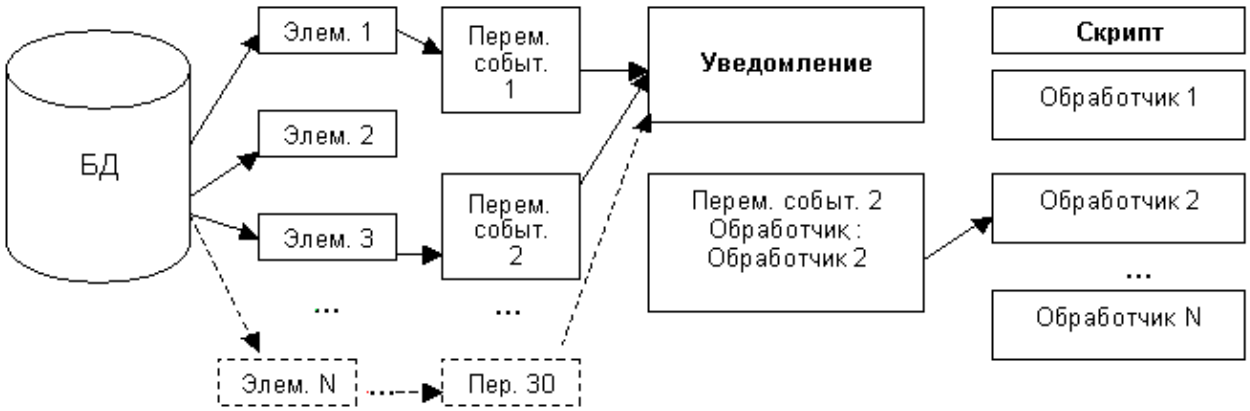
Событие **Уведомление** реагирует на изменение значений переменных базы данных. Это событие кроме общих свойств – обработчик и имя события – имеет дополнительные свойства: переменные события, за которыми осуществляется слежение, и их количество.




В поле **Кол-во переменных** указывается количество переменных события (диапазон от 0 до 30). После того как количество указано в Окне свойств отображается список переменных. Справа от каждой переменной располагается поле для выбора ссылки на базу данных.




Принцип работы события **Уведомление** прост: при открытии мнемосхемы, после инициализации скрипта начинается отслеживание изменений в элементах базы данных, на которые ссылаются переменные события. Если в какой-либо переменной (хотя бы в одной) происходят изменения (изменилось значение), то выполняется код из процедуры-обработчика данного события. Схематически работу события можно представить так:




Переменные события «Уведомление» ссылаются на какие-либо элементы из базы данных. При изменении значения в одном из таких элементов скрипту приходит уведомление об этом, и выполняется код из сопоставленного этому событию обработчика.

**ВНИМАНИЕ!!!**

**Событие «Уведомление» лишь информирует скрипт (через вызов обработчика), что значение элемента базы данных изменилось, но не предоставляет информацию о том чему равно это значение. Другими словами не существует возможности из скрипта, напрямую, обратится к переменным события («Переменная 1», «Переменная 2» и т.д.). Для получения доступа к элементу базы данных, на который ссылается переменная события, необходимо в скрипте объявить БД-переменную, задаваемую ссылкой, и «привязать» ее к соответствующему элементу базы данных.**

### 12.10.3.3 Событие «Сообщение»

Иногда необходимо сообщить одному скрипту о том, что в другом произошло какое-либо событие (например, переменная достигла определенного значения). Для этого предусмотрен механизм посылки скриптами сообщений.

**ВНИМАНИЕ!!!**

**Отсылать сообщения могут скрипты любого вида, а принимать только «Скрипты на мнемосхему».**

Событие **Сообщение** обеспечивает прием сообщений в «Скрипте на мнемосхему».

Для того чтобы послать сообщение необходимо вызвать встроенную функцию **SendMessage** (смотрите описание в разделе 12.12 «Встроенные функции»). У этой функции всего один параметр типа **Variant**, который должен содержать строку с сообщением.

Например:

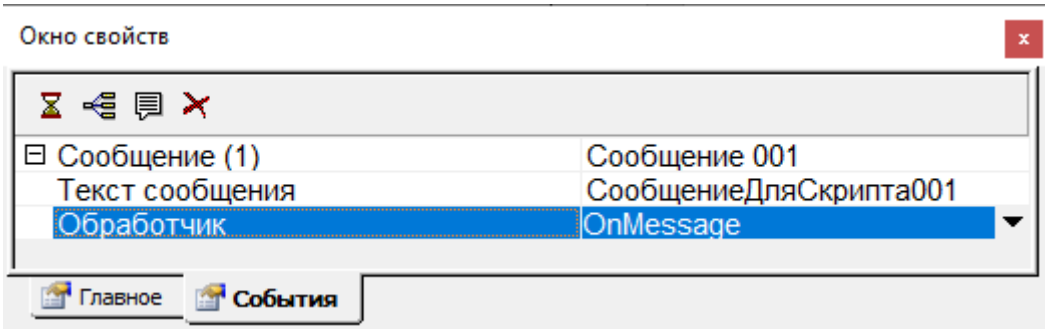
```
SendMessage "СообщениеДляСкрипта001"
```

или

```
Dim msg
msg = "СообщениеДляСкрипта001"
SendMessage msg
```

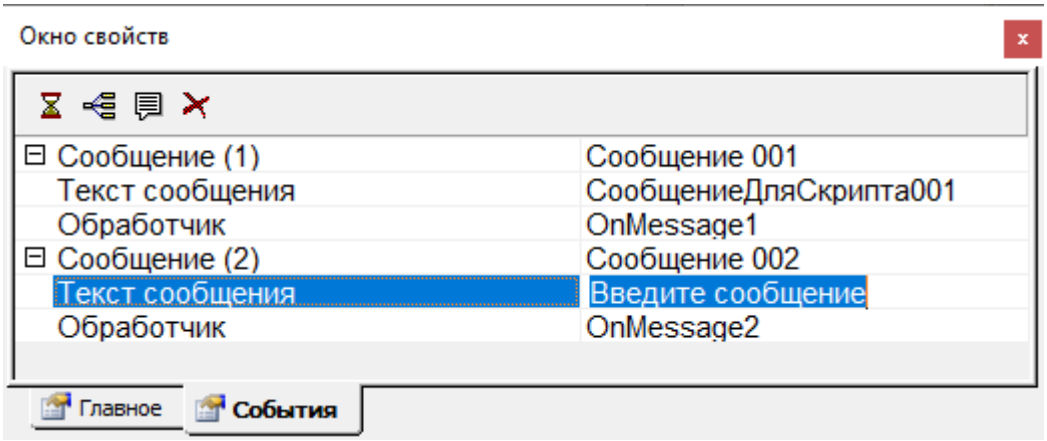
Строка с сообщением может содержать любой текст. Будет ли это осмысленное предложение, словосочетание, либо просто произвольный набор символов, или же один символ – не важно.

Для приема сообщения необходимо создать событие **Сообщение** и в параметре **Текст сообщения** указать сообщение, которое должно обрабатываться этим событием.




Принцип работы события **Сообщение** следующий: когда какой-либо скрипт вызывает функцию **SendMessage**, всем «Скриптам на мнемосхему» рассылается сообщение, содержащее текст, указанный в параметре **SendMessage**. Если открыто несколько мнемосхем, которым назначен **Скрипт на мнемосхему**, то сообщение приходит каждому такому скрипту. «Скрипт на мнемосхему» просматривает все свои события типа **Сообщение** и сравнивает значение параметра **Текст сообщения** с текстом сообщения, отправленного функцией **SendMessage**. Если тексты совпадает (регистр символов не учитывается), то вызывается код скрипта из назначенного этому событию обработчика.

Например, «скрипт на мнемосхему» обрабатывает два события **Сообщение**




Скрипт на мнемосхему имеет следующий вид:

```
<<@Текст>>
Sub OnMessage1 ()
  @Текст = "Пришло сообщение 1"
End Sub
Sub OnMessage2 ()
  @Текст = "Пришло сообщение 2"
End Sub
Sub OnFinishScript ()
  @Текст = "Выполнение завершено"
End Sub
```

Допустим, что некоторый скрипт выполняет следующую последовательность операторов:

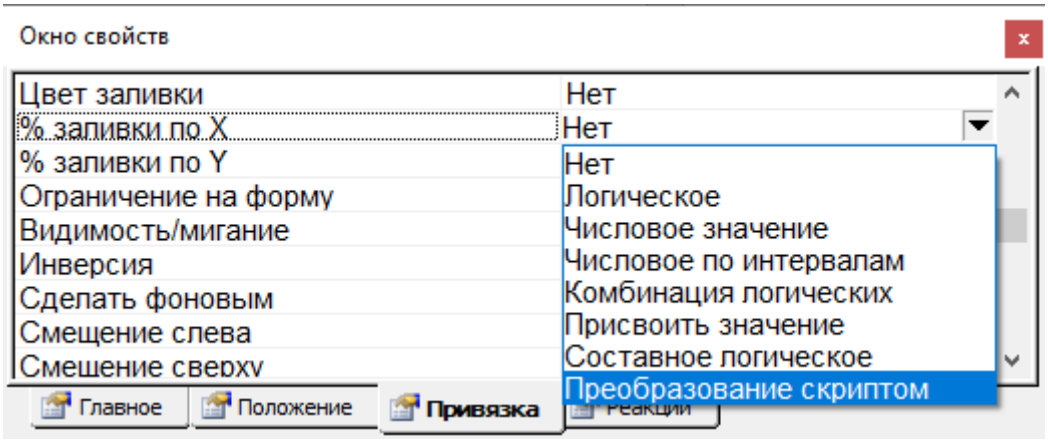
```
...
SendMessage "СообщениеДляСкрипта001"
...
SendMessage "Введите сообщение"
...
SendMessage "Действие3"
...
```

Тогда после вызова **SendMessage "СообщениеДляСкрипта001"** в «скрипте на мнемосхему» будет выполнена процедура **OnMessage1**. После вызова **SendMessage "Введите сообщение"** выполняется процедура **OnMessage2**. Вызов функции **SendMessage "Действие3"** указанный «скрипт на мнемосхему» проигнорирует, так как ни в одном из его событий **Сообщение** не предусмотрена обработка сообщения с текстом **Действие3**.

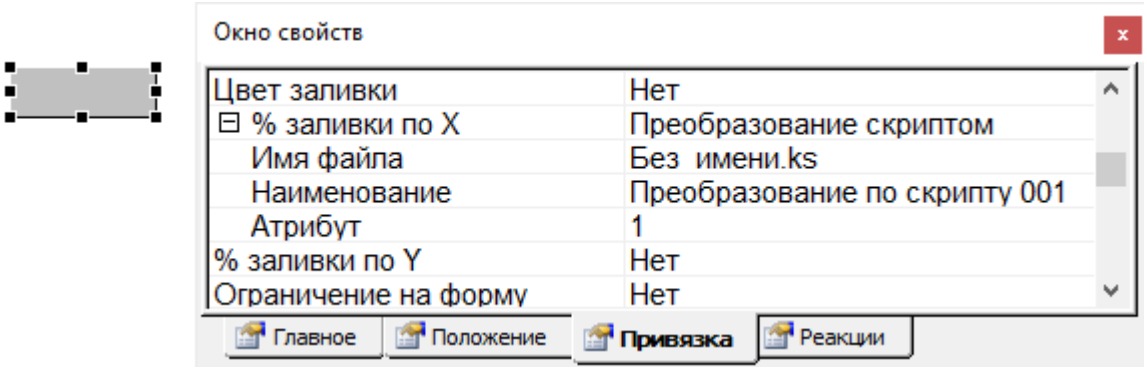
### 12.10.4 Функция «Преобразование скриптом»

Функция **Преобразование скриптом,** как и стандартные функции преобразования назначается на закладке **Привязка** в Окне свойств динамического элемента.

Особенностью данной функции преобразования является способность управлять значением атрибута, на который она назначена, с помощью скрипта.




В Окне свойств из списка доступных преобразований необходимо выбрать **Преобразование скриптом**. В поле **Имя файла** нужно указать имя файла со скриптом.




Управление значением атрибута осуществляется через предопределенную БД-переменную **%Атрибут**. При записи в эту переменную какого-либо значения это значение записывается и в атрибут, на который назначена функция преобразования. Переменная **%Атрибут** ведет себя подобным образом только в функции **Преобразование скриптом**. В остальных случаях (**Скрипт на мнемосхему**, функция реакции **Выполнить скрипт**) она представляет собой обычную БД-переменную, задаваемую значением. Переменная **%Атрибут** автоматически добавляется в область видимости скрипта, даже если в скрипте нет ни одного упоминания о ней.

Принцип работы функции **Преобразование скриптом** следующий: при открытии мнемосхемы выполняются все стандартные операции по загрузке скриптов. Затем выполняется код расположенный в *секции исполнения*.

**ВНИМАНИЕ!!!**

**Секция исполнения начинается с двух открывающих фигурных скобок и заканчивается двумя закрывающими. Между скобками не должно быть пробелов.**

Секция исполнения является аналогом процедуры-обработчика и выглядит так:

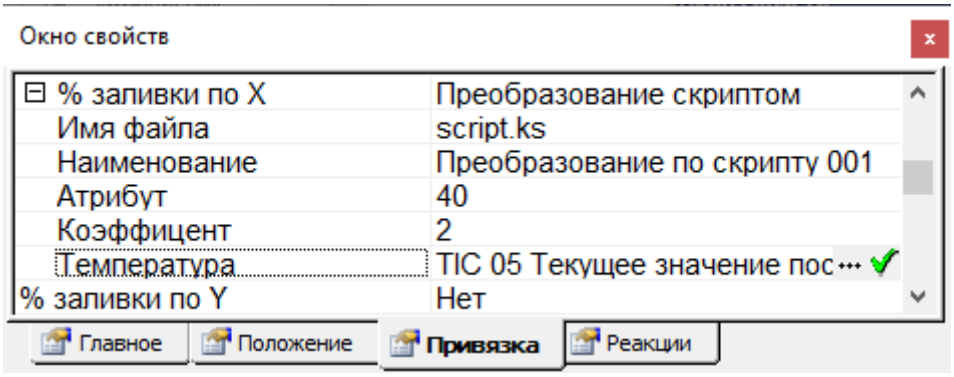
```
{{
  <операторы>          'Что-то делаем в секции исполнения
}}
```

После выполнения операторов секции исполнения скрипт переходит в состояние ожидания событий.

Для функции **Преобразование скриптом** существует только одно единственное событие: изменение значения хотя бы одной БД-переменной скрипта (кроме переменной **%Атрибут**). Другими словами мы предполагаем, что переменная **%Атрибут** (а значит и сам атрибут, на

который назначена функция преобразования) каким-то образом зависит от находящихся в скрипте БД-переменных. При изменении одной из них необходимо обновить значение атрибута, что и должно быть сделано в секции исполнения. Графически это можно представить так:


БД-переменные скрипта, задаваемые значением, можно использовать в качестве входных параметров скрипта. В этом случае в Окне свойств заранее устанавливаются значения этих переменных.




Атрибуту **Текст** графического примитива назначен следующий скрипт:

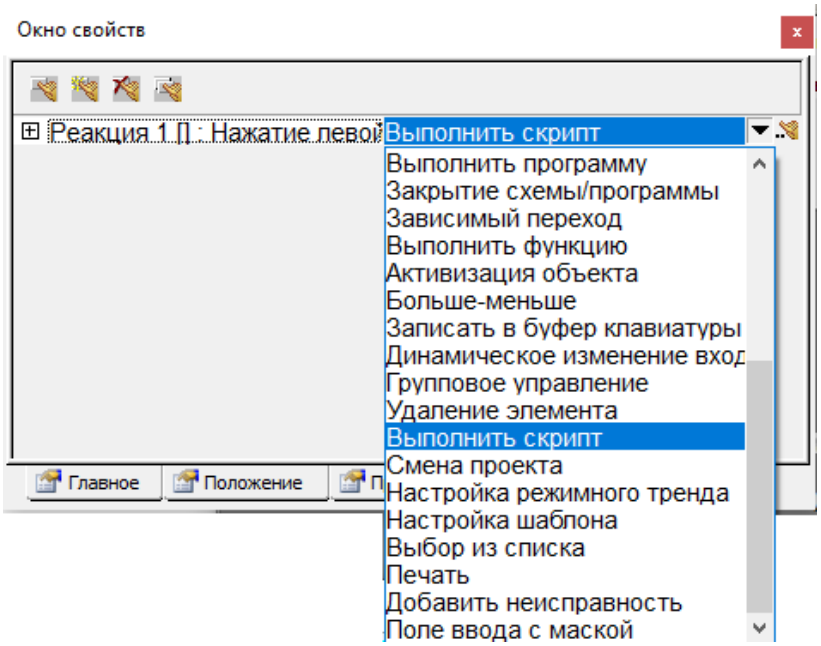
```
{{
  %Атрибут = %Коэффицент * @Температура
}}
Sub OnFinishScript ()  'Выполните завершающие действия в этой процедуре
  ...
End Sub
```

После выполнения кода секции исполнения значение переменной **%Атрибут** будет равно 40. БД-переменную **%Коэффициент** в данном случае можно рассматривать как входной параметр скрипта. В Окне свойств задается значение этого параметра равное 2.

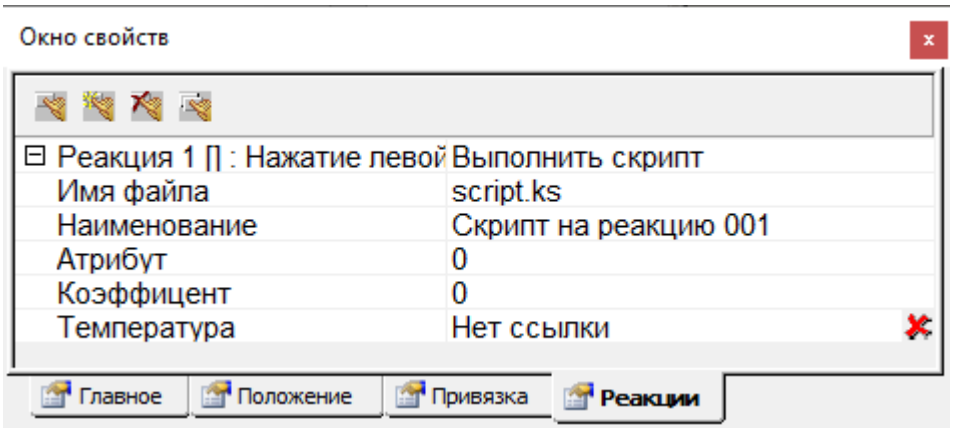
Таким образом, можно назначать один и тот же скрипт на различные атрибуты и присваивая входным параметрам различные значения, получать в результате разные значение атрибута.

### 12.10.5 Функция реакция «Выполнить скрипт»

В Окне свойств динамического элемента на закладке **Реакции** из списка событий выбирается нужное и для него назначается функция реакции **Выполнить скрипт**.




В поле **Имя файла** необходимо указать имя файла скрипта.




Если в скрипте есть БД-переменные, задаваемые ссылкой, то их необходимо «привязать» к БД.

Принцип работы реакции **Выполнить скрипт**: при возникновении определенного события, с котором сопоставлена данная функция реакции, выполняется код секции исполнения указанного скрипта.

### 12.11 Библиотеки скриптов

Скрипты могут включать в себя одну или несколько библиотек скриптов.

**Библиотека скриптов** представляет собой текстовый файл с расширением **kl** и содержит текст скрипта (это могут быть объявления процедур, функций, переменных, классов; операторы и другие элементы).

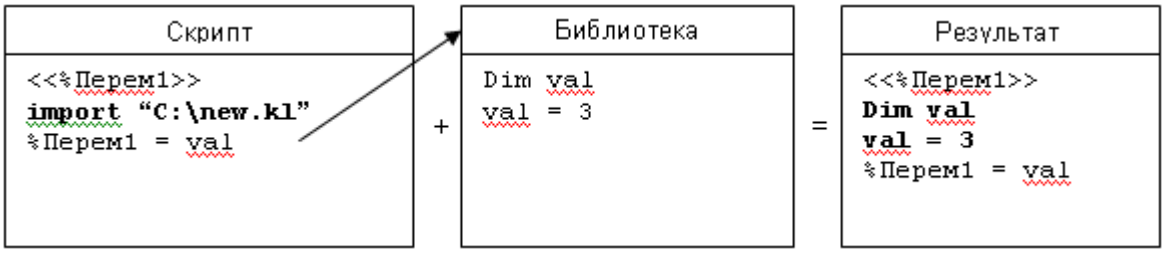
Имя файла библиотеки может быть либо абсолютным, либо относительным. Библиотеки могут включать в себя другие библиотеки. Скрипт не может два раза включить в себя библиотеку с одним и тем же именем.

Для подключения библиотеки к скрипту используется следующий оператор:

**import "Имя файла библиотеки"**

Оператор **import** должен быть единственной инструкцией в строке.

При включении в скрипт библиотеки строка с оператором **import** заменяется текстом из подключаемой библиотеки. Например:




## 12.12 Встроенные функции

Язык скриптов системы КРУГ-2000 кроме стандартного набора функций языка VBScript, включает себя несколько групп дополнительных функций, называемых **встроенными**. Эти функции используют такие же соглашения о синтаксисе, как и функции VBScript.

Далее приводится описание встроенных функций (по группам) и примеры их использования.

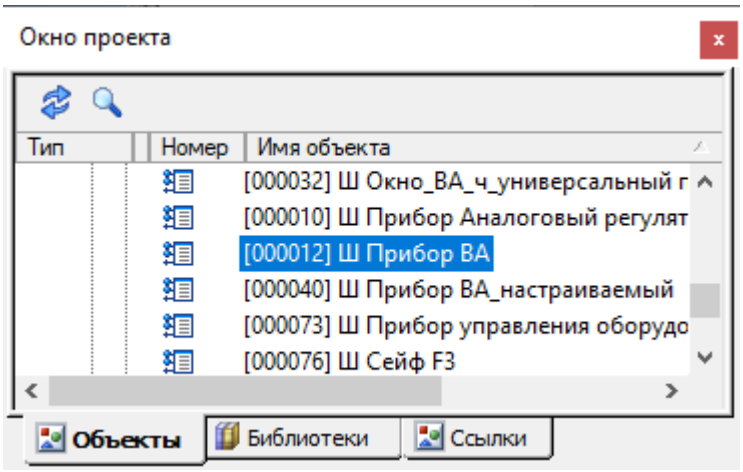
### 12.12.1 Работа с прибором

Таблица 12.12.1. Функции группы «Работа с прибором»

| Функция | Назначение |
|---|---|
| **CreateDevice «Имя прибора»** | Создает прибор с заданным именем |
| **DestroyDevice «Имя прибора»** | Удаляет прибор с заданным именем |
| **DeviceOpen («Имя прибора», позиция X, позиция Y)** | Открывает прибор с заданным именем в заданном месте экрана. Возвращает идентификатор прибора (ИП) |
| **DeviceClose ИП** | Закрывает прибор с заданным идентификатором |
| **DevicePosition ИП, позиция X, позиция Y** | Помещает прибор с заданным идентификатором в заданное место экрана |
| **SetDeviceInput «Имя прибора», «Имя входа», «Имя БД-переменной»** | Сопоставляет входы прибора БД-переменным. Имя БД-переменной задается в кавычках, переменная должна быть задана как ссылка на БД. Например: **@Переменная_1** |
| **IsDeviceCreated («Имя прибора»)** | Возвращает **True**, если прибор создан, иначе **False** |
| **IsDeviceExist ИП** | Проверяет, существует ли прибор с заданным идентификатором. Возвращает **True**, если прибор существует (открыт), иначе – **False** |
| **GetDeviceInputsCount «Имя прибора»** | Возвращает количество входов прибора |

Для работы с прибором через скрипт, первоначально нужно вызвать функцию **CreateDevice**, которая проводит начальную инициализацию, необходимую для работы прибора. Параметром функции **CreateDevice** является строка с именем прибора.

После этого необходимо задать входы для созданного прибора. Это делает функция **SetDeviceInput**. Первый параметр – строка с именем прибора, второй – имя входа прибора. Третий параметр – это строка с именем БД-переменной, задаваемой ссылкой, которая «привязывается» к данному входу.




Например:

```
<<@Вход1>>
CreateDevice "Ш Прибор ВА"
SetDeviceInput "Прибор ВА", "ВА", "@Вход1"
```

Если БД-переменная в третьем параметре функции **SetDeviceInput** больше нигде в скрипте не используется, необходимо предварительно объявить эту переменную в секции объявления БД-переменных, так как в **SetDeviceInput** она передается как строка и не распознается транслятором как БД-переменная.

БД-переменные должны быть сопоставлены всем входам прибора. Если у прибора пять входов, то для каждого из них необходимо вызвать функцию **SetDeviceInput**. Можно узнать количество входов прибора, вызвав функцию **GetDeviceInputsCount**, указав в параметре имя прибора.

**ВНИМАНИЕ!!!**

**Создание прибора функцией CreateDevice и сопоставление входам прибора соответствующих переменных функцией SetDeviceInput должно выполняться в скрипте один раз. Поэтому, вызовы этих функций необходимо делать в области видимости скрипта – не в процедурах-обработчиках и не в секции исполнения!**

После того как входы «привязаны» для отображения прибора необходимо вызвать функцию **DeviceOpen**. Первый параметр функции это строка с именем прибора, второй и третий это координаты **x** и **y** правого верхнего угла прибора относительно мнемосхемы. Так как можно открыть одновременно несколько приборов с одинаковым именем, для возможности доступа к конкретному прибору функция **DeviceOpen** возвращает идентификатор открытого прибора (**ИП**).

Функция **DeviceOpen** может открыть только тот прибор, на который есть соответствующий переход. Например:

```
Dim devId
devId = DeviceOpen ("Ш Прибор ВА", 0, 0)
```

С помощью идентификатора прибора можно менять положение прибора на экране, закрыть прибор и узнать существует ли прибор с таким идентификатором.

Например:

```
<<@Вход1>>
Dim devId
CreateDevice "Ш Прибор ВА"
SetDeviceInput " Ш Прибор ВА", "ВА", "@Вход1"
{{
  if @Var = 0 Then
```

```
        devId = DeviceOpen("Ш Прибор ВА", 0, 0)
    end if
    if @Var = 10 And IsDeviceExist(devId)Then
        DevicePosition devId, 100, 100
    end if
    if @Var = 100 Then
        DeviceClose devId
    end if
}}
```

Функция **IsDeviceCreated** проверяет, создан ли прибор, т.е. была ли для прибора вызвана функция **CreateDevice** и не был ли он уничтожен функцией **DestroyDevice**. Параметром функции **IsDeviceCreated** является строка с именем прибора.

Функция **DestroyDevice** уничтожает прибор с указанным именем, созданный функцией **CreateDevice** – **не шаблон прибора!** После вызова этой функции, доступ к прибору с именем, указанным в параметре функции **DestroyDevice** будет закрыт, при этом зарываются все копии прибора открытые функцией **DeviceOpen**.

Функция **DestroyDevice** обычно вызывается при завершении скрипта в процедуре **OnFinishScript**. Например:

```
Sub OnFinishScript ()
  if IsDeviceCreated ("Ш Прибор ВА") Then
      DestroyDevice " Ш Прибор ВА"
  end if
End Sub
```

### 12.12.2 Изображение

Таблица 12.12.2. Функции группы «Изображение»

| Функция | Назначение |
|---|---|
| **CreatePicture ("Имя файла", позиция X, позиция Y, цвет фона)** | Создает изображение по заданным параметрам. Возвращает идентификатор изображения (ИИ). Поддерживает форматы: **gif, jpeg, bmp, wmf, ico, cur** |
| **DestroyPicture ИИ** | Уничтожает изображение с заданным идентификатором |
| **PicturePosition ИИ, позиция X, позиция Y** | Перемещает изображение в точку с заданными координатами |
| **ShowPicture ИИ** | Показать изображение |
| **HidePicture ИИ** | Скрыть изображение |
| **IsPictureVisible ИИ** | Возвращает **True**, если картинка видима, иначе – **False** |

Работа с изображением в скриптах начинается с вызова функции **CreatePicture**. Первый параметр функции – строка с именем файла изображения. Имя файла может быть как абсолютным, так и относительным. Второй и третий параметры это координаты левого верхнего угла изображения относительно мнемосхемы. Четвертый параметр это цвет фона изображения, он актуален только в случае, если изображение поддерживает прозрачный цвет фона. Последующая работа с созданным изображением осуществляется через идентификатор, который возвращает функция **CreatePicture**. Например:

```
Dim pId
pId = CreatePicture ("f:\project\tank1_g.bmp", 0, 0, RGB (0,0,255))
```

Изначально изображение невидимо. Для того чтобы сделать его видимым необходимо вызвать функцию **ShowPicture**. Параметром функции является идентификатор изображения.

Для того чтобы спрятать изображение, предусмотрена функция **HidePicture**.

Функция **PicturePosition** изменяет положение изображения на экране. Например:

```
Dim pId
pId = CreatePicture ("f:\project\tank1_g.bmp", 0, 0, RGB (0,0,255))
ShowPicture pId
{{
  if IsPictureVisible (pId) Then
      PicturePosition pId, @PosX, @PosY
  end if
}}
Sub OnFinishScript ()
  DestroyPicture pId
End Sub
```

Функция **DestroyPicture** уничтожает изображение с заданным идентификатором. После того как изображение уничтожено, обратится к нему по идентификатору невозможно.

### 12.12.3 События

Таблица 12.12.3. Функции группы «События»

| Функция | Назначение |
|---|---|
| **EnableTimer «Имя таймера»** | Разрешает работу таймера с заданным именем |
| **DisableTimer «Имя таймера»** | Запрещает работу таймера с заданным именем |
| **SetTimerInteval «Имя таймера», Интервал в мс** | Задает интервал для работы таймера |
| **EnableMessage «Имя сообщения»** | Разрешает обработку сообщения с заданным именем |
| **DisableMessage «Имя сообщения»** | Запрещает обработку сообщения с заданным именем |
| **SendMessage «Текст сообщения»** | Посылает сообщение |
| **EnableAdvise «Имя уведомления»** | Разрешает обработку уведомления с заданным именем |
| **DisableAdvise «Имя уведомления»** | Запрещает обработку уведомления с заданным |

| Функция | Назначение |
|---|---|
| | именем |

**ВНИМАНИЕ!!!**

**Функции обработки событий, кроме SendMessage могут быть вызваны только из скрипта на мнемосхему**

Из функций для работы с событиями можно выделить функции, которые разрешают и запрещают работу событий. Так, например, вызвав функцию **DisableTimer** с параметром **«Таймер_001»** мы запрещаем в данном скрипте на мнемосхему обработку события **Таймер** с именем **Таймер_001**. Функции разрешения/запрешения можно вызывать только в скрипте на мнемосхему и применительно к событиям этого скрипта. Вызов функции **EnableTimer** с параметром **«Таймер_001»** приведет к возобновлению работы события **Таймер** с именем **Таймер_001**.
Например:

```
' Скрипт на мнемосхему
Sub OnMsgEnableTimer ()
   EnableTimer "Таймер_001"
End Sub
Sub OnMsgDisableTimer ()
   DisableTimer "Таймер_001"
End Sub
'Выполните завершающие действия в этой процедуре
Sub OnFinishScript ()
   MsgBox "Выполнение завершено"
End Sub
```

Функция **SetTimerInterval** устанавливает новый интервал для события **Таймер**. Первый параметр – это строка с именем таймера, второй – интервал выполнения. Функция **SetTimerInterval** должна вызываться из скрипта на мнемосхему и применительно к событиям **Таймер** данного скрипта.
Например:

```
Sub OnMsgChangeInterval_100 ()
   'Установить интервал 100 мс
   SetTimerInterval "Таймер_001", 100
End Sub
Sub OnMsgChangeInterval_200 ()
   'Установить интервал 200 мс
   SetTimerInterval "Таймер_001", 200
End Sub
```

Функция **SendMessage** рассылает сообщение всем открытым скриптам на мнемосхему. Параметром функции является строка с текстом сообщения. Функция **SendMessage** может

быть вызвана из любого скрипта (функция **Преобразование скриптом**, реакции **Выполнить скрипт** и **Скрипт на мнемосхему**).
Например:

```
<<@Var>>
{{
  if @Var = 20 Then
      SendMessage "Значение = 20"
  end if
}}
Sub OnFinishScript ()
  @Var = 0
End Sub
```

### 12.12.4 Мнемосхема

Таблица 12.12.4. Функции группы «Мнемосхема»

| Функция | Назначение |
|---|---|
| **CloseProgram** | Закрывает приложение (ГИ или ГД в зависимости от того, откуда вызывается). Если в приложении есть не сохраненные данные, то программа предложит их сохранить. |
| **MnemoOpen «Имя мнемосхемы»** | Открывает мнемосхему с заданным именем |
| **MnemoOpenById Идентификатор мнемосхемы** | Открытие мнемосхемы по заданному идентификатору (номер мнемосхемы в проекте) |
| **MnemoClose** | Закрывает мнемосхему из которой вызван скрипт |
| **MnemoCloseByName “Имя мнемосхемы”** | Закрывает мнемосхему с указанным именем |
| **MnemoCloseById Идентификатор мнемосхемы** | Закрывает мнемосхему по заданному идентификатору (номер мнемосхемы в проекте) |
| **MnemoLocation координата x, координата y** | Устанавливает координаты мнемосхемы |
| **MnemoSize ширина, высота** | Устанавливает размеры мнемосхемы |
| **MnemoLayout «Параметры»** | Устанавливает вид мнемосхемы по заданным параметрам:<br>• **MAXIMIZE»** – во весь экран<br>• **«CASCADE»** – открытые мнемосхемы располагаются каскадом<br>• **«HORIZONTAL»** – горизонтальное расположение<br>• **«VERTICAL»** – вертикальное расположение<br>• **«MINIMIZE»** – сворачивает мнемосхему<br>• **«RESTORE»** – восстанавливает предыдущее состояние<br>• **«ACTIVATE»** – сделать мнемосхему активной |

| Функция | Назначение |
|---|---|
| | • **«ICONARRANGE»** – сортировка свернутых мнемосхем |
| **GetMnemoX** | Возвращает координату **x** мнемосхемы |
| **GetMnemoY** | Возвращает координату **y** мнемосхемы |
| **GetMnemoHeight** | Возвращает высоту мнемосхемы |
| **GetMnemoWidth** | Возвращает ширину мнемосхемы |
| **UpdateScreen** | Обновляет содержимое мнемосхемы |
| **DesktopOpen “Имя рабочего стола”** | Открывает рабочий стол с заданным именем |
| **DesktopOpenById Идентификатор рабочего стола** | Открывает рабочий стол с заданным идентификатором (номер рабочего стола в проекте) |
| **RunApplication “Имя файла”** | Запускает приложение по имени файла (либо **exe**, либо другой тип, если ему сопоставлено приложение для запуска). Возвращает идентификатор запущенного приложения, либо **0** если идентификатор не доступен. Например: **AppId = RunApplication “Имя файла”** |
| **RunApplicationParams “Имя файла”, “Параметры запуска”** | Запускает приложение с параметрами |
| **CloseApplication AppId** | Закрывает приложение по заданному идентификатору. Эквивалентно закрытию приложения нажатием на клавишу  . |
| **TerminateApplication AppId** | Уничтожает приложение по заданному идентификатору без сохранения каких-либо данных этого приложения |
| **WaitForClosing AppId, Интервал мс** | Ожидает завершения приложения с заданным идентификатором в течении заданного интервала времени. Интервал задается в миллисекундах. Возвращаемое значение: **True** – если приложение завершилось, **False** – если истек отведенный интервал времени, **1** – если произошла ошибка. Например: **result = WaitForClosing AppId, Интервал мс** |

Функция **MnemoOpen** открывает мнемосхему с заданным именем или идентификатором. Параметром функции является строка с именем мнемосхемы или её номером. Функция **MnemoOpen** может открыть только ту мнемосхему, на которую есть соответствующий переход.
Например:

```
Sub OnMnemoOpen ()
  MnemoOpen "Мнемосхема 003"
End Sub
```

Функция **MnemoClose** закрывает мнемосхему. Закрывается та мнемосхема, к которой «привязан» скрипт, вызвавший функцию **MnemoClose**. Функцию **MnemoClose** запрещено использовать при инициализации скрипта и в функции **Преобразование скриптом**.
Например:

```
MnemoClose          'Ошибка – инициализация скрипта
Sub OnMnemoClose ()
  MnemoClose        'Правильно – надо вызывать функцию в обработчике
End Sub
```

Функции **MnemoLocation**, **MnemoSize** и **MnemoLayout** управляют расположением мнемосхемы в окне приложения.
Например:

```
Sub SetMnemoParam ()
  MnemoLocation 100, 200
  MnemoSize 500, 500
  MnemoLayout "VERTICAL"
End Sub
```

Функции **GetMnemoX**, **GetMnemoY**, **GetMnemoHeight**, **GetMnemoWidth** – возвращают параметры мнемосхемы координаты и размеры мнемосхемы.
Например:

```
Dim x, y, height, width
Sub GetMnemoParam ()
  x = GetMnemoX
  y = GetMnemoY
  height = GetMnemoHeight
  width = GetMnemoWidth
End Sub
```

Функция **UpdateScreen** – обновляет содержимое мнемосхемы (перерисовывает мнемосхему). Пользоваться этой функцией нужно только при необходимости. Слишком частое обновление может вызвать сильную загрузку процессора

### 12.12.5 Мышь и клавиатура

Таблица 12.12.5. Функции группы «Мышь и клавиатура»

| Функция | Назначение |
|---|---|
| **MouseAction действие, клавиша** | Имитирует события мыши.<br>Действие:<br>• 0 – клавиша нажата,<br>• 1 – клавиша отжата,<br>• 2 – одинарное нажатие,<br>• 3 – двойное нажатие<br>Клавиша:<br>• 0 – левая клавиша,<br>• 1 – средняя клавиша, |

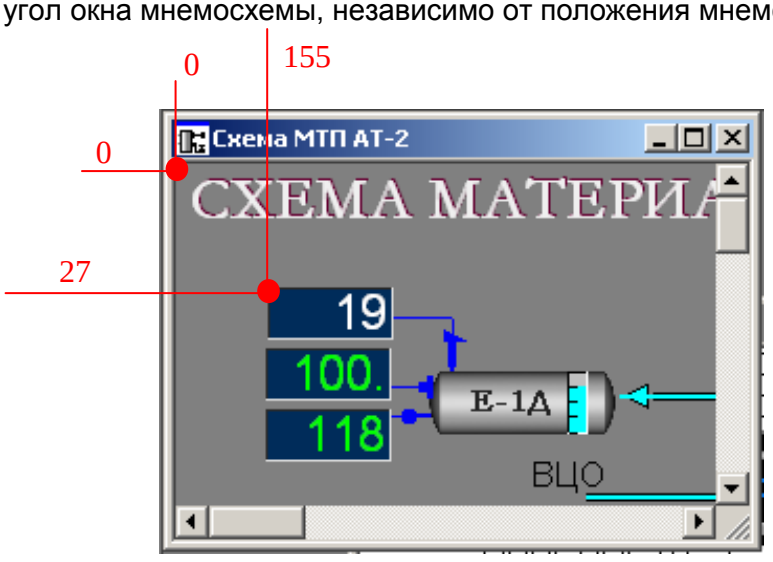
| Функция | Назначение |
|---|---|
| | • 2 – правая клавиша. |
| **MousePosition координата x, координата y** | Устанавливает координаты мыши |
| **GetMouseX** | Возвращает координату **x** курсора мыши |
| **GetMouseY** | Возвращает координату **y** курсора мыши |
| **ConvertXtoMnemo (координата x окна мнемосхемы)** | Преобразует координату **x** окна мнемосхемы в координату **x** мнемосхемы |
| **ConvertYtoMnemo (координата y окна мнемосхемы)** | Преобразует координату **y** окна мнемосхемы в координату **y** мнемосхемы |
| **ConvertXtoWindow (координата x мнемосхемы)** | Преобразует координату **x** мнемосхемы в координату **x** окна мнемосхемы. |
| **ConvertYtoWindow (координата y мнемосхемы)** | Преобразует координату **y** мнемосхемы в координату **y** окна мнемосхемы |
| **KeyboardAction код клавиши** | имитирует нажатие клавиши с заданным кодом. Список кодов клавиш приведен в **Приложении А.** |

Функции по работе с мышью и клавиатурой предназначены для имитации событий этих устройств, таких как нажатие/отжатие клавиши, перемещение курсора и т.д. Первая в этом списке стоит функция **MouseAction**, которая имитирует события мыши. Первый параметр функции это действие. Второй параметр указывает клавишу, для которой это действие выполняется. Например, для имитации клика (одинарное нажатие) левой клавишей мыши необходимо выполнить: **MouseAction 2, 0**

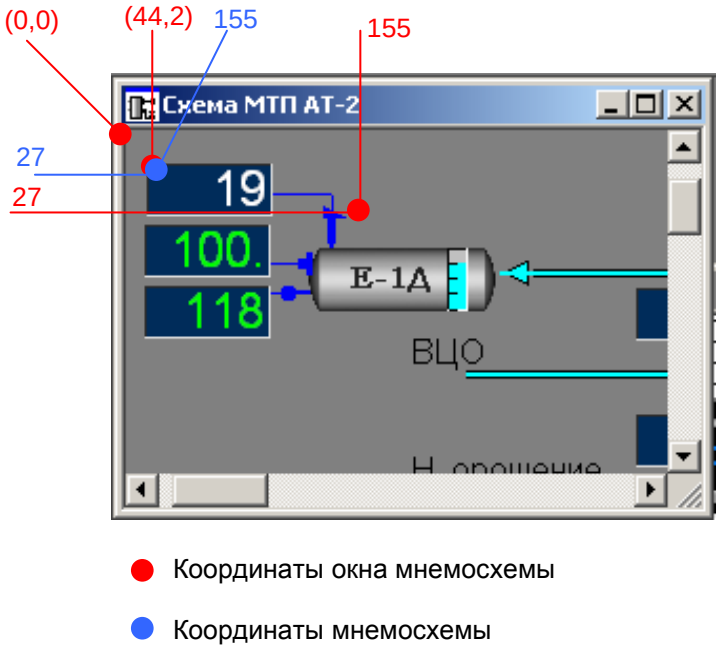
Для того чтобы переместить курсор мыши в заданную точку экрана, необходимо вызвать функцию **MousePosition**. Первый параметр у нее это координата **x**, второй координата **y**. Координаты задаются относительно окна мнемосхемы.

В чем отличие координат окна мнемосхемы от координат мнемосхемы?

Для координат окна мнемосхемы точка с координатами (0, 0) – это всегда левый верхний угол окна мнемосхемы, независимо от положения мнемосхемы внутри окна.




В случае с координатами мнемосхемы, наоборот. Точка с координатами (0, 0), а вместе с ней и все точки мнемосхемы при скроллинге смещаются.




Функции **GetMouseX** и **GetMouseY** возвращают положение курсора мыши относительно окна мнемосхемы. Первая функция возвращает координату **x,** вторая – координату **y**.

К группе функций для работы с мышью, добавлены четыре вспомогательные функции для преобразования координат мнемосхемы в координаты окна мнемосхемы, и обратно. Это: **ConvertXtoMnemo**, **ConvertYtoMnemo**, **ConvertXtoWindow** и **ConvertYtoWindow**. Каждая из функций имеет один параметр. Для функций **ConvertXtoMnemo** и **ConvertYtoMnemo** это соответственно координата **x** окна мнемосхемы и координат **y** окна мнемосхемы. Результатом функции **ConvertXtoMnemo** является координата **x** мнемосхемы, которая соответствует координате **x** окна мнемосхемы. То же самое для координаты **y** в функции **ConvertYtoMnemo**. Для функций **ConvertXtoWindow** и **ConvertYtoWindow** наоборот – на входе координата мнемосхемы, на выходе координата окна мнемосхемы.

Для работы с клавиатурой предусмотрена только одна функция **KeyAction**. Результатом выполнения этой функции является имитация быстрого нажатия/отжатия клавиши клавиатуры.

### 12.12.6 Звук

Таблица 12.12.6. Функции группы «Звук»

| Функция | Назначение |
|---|---|
| **PlayFile «Имя файла»** | Проигрывает звуковой файл в формате wav |
| **StopSound** | Прекращает проигрывание файла, запущенного функцией **PlayFile** |

Работа со звуком в скриптах реализуется через использование двух встроенных функций: **PlaySound** и **StopSound**. Первая проигрывает файл формата **wav**, вторая прекращает воспроизведение. Функция **PlaySound** не начнет проигрывание нового файла, если в данный момент проигрывается что-либо еще. Для воспроизведения файла, в любом случае, необходимо предварительно вызвать функцию **StopSound**.
Например:

```
{{
  StopSound
  PlayFile "C:\Project\alarm.wav"
}}
Sub OnFinishScript ()
  StopSound
End Sub
```

Имя звукового файла в функции **PlaySound** может быть задано как абсолютным, так и относительным.

### 12.12.7 Отладка

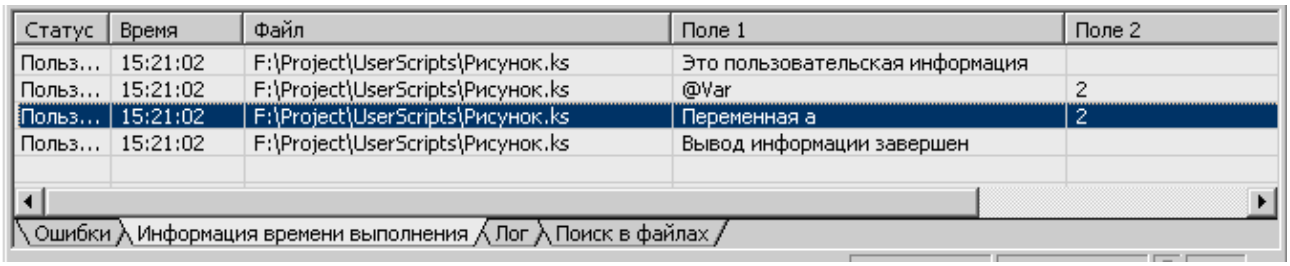
Таблица 12.12.7. Функции группы «Отладка»

| Функция | Назначение |
|---|---|
| **Trace значение** | Отображает в окне информация времени выполнения редактора скриптов параметр функции в поле **Поле 1** |
| **Trace2 значение1**, **значение2** | То же, что и **Trace**, но отображаются два параметра в полях **Поле 1** и **Поле *2*** |

В скриптах системы КРУГ-2000 предусмотрены функции **Trace** и **Trace2** для вывода пользовательской информации в Редакторе скриптов.
Например:

```
<<@Var>>
Dim a, b
Trace "Это пользовательская информация"
@Var = 2
Trace2 "@Var", @Var
a = "Переменная a"
b = @Var
Trace2 a, b
Trace "Вывод информации завершен"
```

После выполнения приведенного выше скрипта, в Редакторе скриптов на закладке «Информация времени выполнения» появятся следующая информация.




**ВНИМАНИЕ!!!**

**Функции Trace и Trace2 предназначены для работы только в Генераторе динамики, в Графическом интерфейсе они ничего не выполняют.**

### 12.12.8 Система

Таблица 12.12.8. Функции группы «Система»

| Функция | Назначение |
|---|---|
| **SendMessageToWindow ("Имя окна", Идентификатор сообщения, Параметр 1,** | Передача сообщений окну по его имени. |

| Функция | Назначение |
|---|---|
| **Параметр 2)** | |

Например:

Сообщение программе с именем окна **TestProgram.** Программа при получении сообщения **WM_USER** с параметром **0x40000038** отображает свое окно настройки в позиции, заданной в параметре 2 – **lParam.**

```
Const WM_USER = 1024                '0x0400
Const ShowSetup = 1073741880        '$40000038
Dim lParam
lParam = 100
Dim result
result = SendMessageToWindow("TestProgram", WM_USER, ShowSetup, lParam)
```

## 12.13 Примеры скриптов

### 12.13.1 Чтение /запись в Excel

Если в ходе работы необходимо создать отчёт в Excel, записать в него данные или забрать данные из уже существующего файла, то можно воспользоваться функциями, описанными в примере ниже.

```
'Переменная БД
<<@Var1>>

'Шаблон отчета
Const FileReport = "C:\Report.xls"

Dim ExcelApp
'Объект Excel
Set ExcelApp = CreateObject("Excel.Application")
'Объект Книга. Файл отчета
Set xlBook = ExcelApp.Workbooks.Open(FileReport)

'Индекс строки для записи
Dim Row
Row = 0

'Чтение из отчета значения в переменную
Sub ReadFromReport ()
  @Var1 = xlBook.Worksheets(1).Range("B1").Offset(1+Row,0).Value
End Sub
```

```
'Запись в отчет текущего значения переменной
Sub WriteToReport ()
  xlBook.Worksheets(1).Range("B1").Offset(0+Row,0).Value = "Значение"
  xlBook.Worksheets(1).Range("B1").Offset(1+Row,0).NumberFormat = "0.00"
  xlBook.Worksheets(1).Range("B1").Offset(1+Row,0).Value = @Var1
End Sub

'Завершающие действия при закрытии скрипта
Sub OnFinishScript ()
  'Закрытие с сохранением
  xlBook.Close True
  ExcelApp.Quit
End Sub
```

### 12.13.2 Отправка e-mail

Пример скрипта:

```
'Подключение библиотек
'import "Имя файла библиотеки"

'Объявление переменных БД:
' <<@Перем1, @Перем2, %Перем1,..., @ПеремN, %ПеремN>>
'<< >>

'Инициализация переменных, описание функций и процедур

'Секция выполнения (используется в реакциях и функциях преобразования)
'{{
…
'}}

Sub SendNotify ()
  Set Mail = CreateObject("CDO.Message")
  Mail.To = "user1@mail.ru"
  Mail.From = "user2@mail.ru"
  Mail.Subject = "Изменение значения переменной"
  Mail.TextBody = "Текст сообщения"
  'Mail.AddAttachment "C:\info.txt"  – если есть вложение
   Mail.Send
End Sub

'Выполните завершающие действия в этой процедуре
Sub OnFinishScript ()
…
```

```
End Sub
```

## 12.14 Система предварительной трансляции скриптов

Для увеличения скорости загрузки скриптов все использующиеся скрипты проекта в оттранслированном виде сохраняются в файле с именем **ИмяПроекта.kgx**, который находится в каталоге проекта.

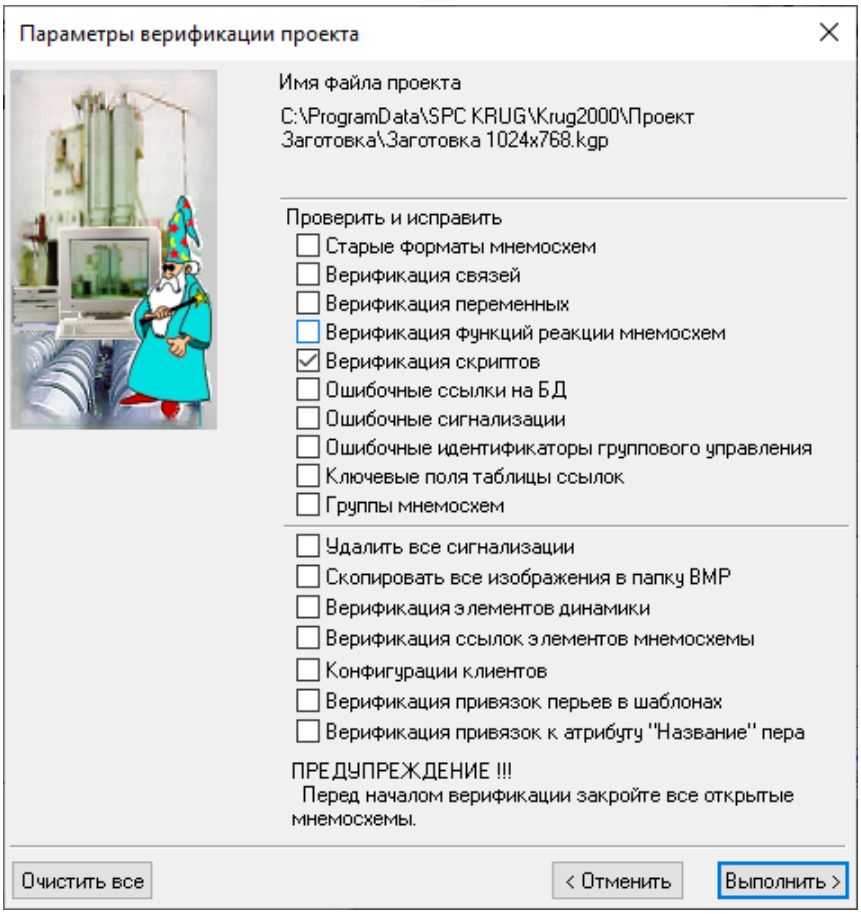
При загрузке файла скрипта проверяется время его последней модификации, если оно не изменилось, то скрипт загружается из **kgx**-файла, если изменилось – скрипт транслируется заново.

Если все скрипты оттранслированы, то для их работы необходим только один единственный **kgx**-файл.

## 12.15 Верификация скриптов

Для уменьшения размера **kgx**-файла и более быстрого поиска в нем оттранслированных скриптов рекомендуется выполнять верификацию.

Для этого следует вызвать окно **Параметры верификации проекта** из меню **Файл/Верификация проекта,** установить признак **Верификация скриптов** и нажать на кнопку **Выполнить**.




## 12.16 Лог файлы

Скрипты записывают информацию об ошибках в лог-файл **ScriptError.log.**

Данный файл сохраняется в каталоге проекта.

# 13 РЕДАКТОР СКРИПТОВ

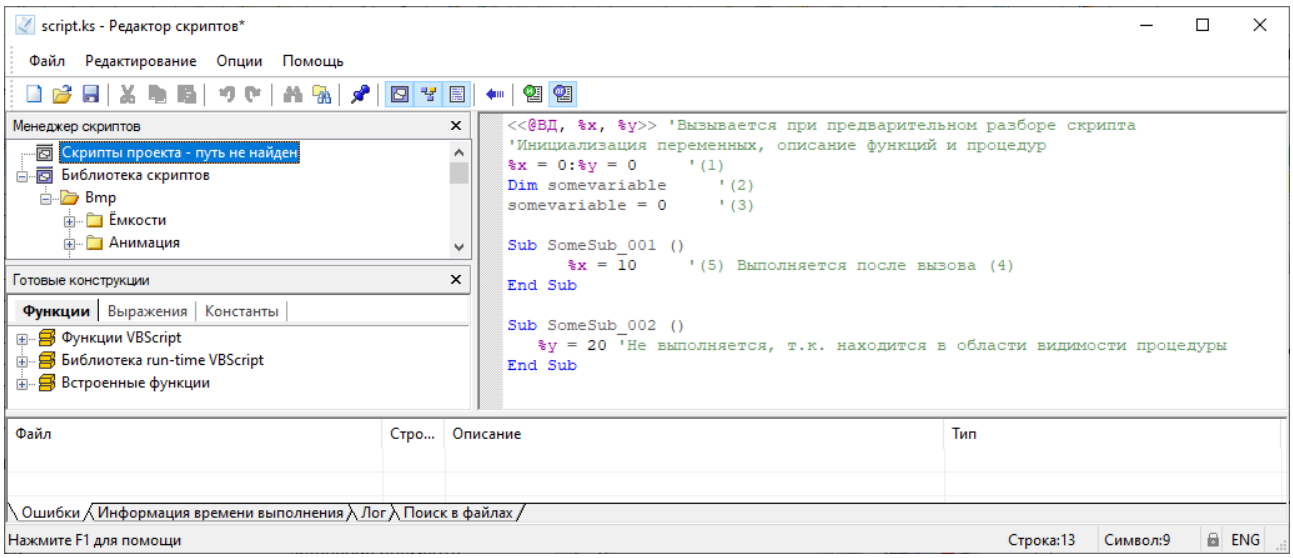
## 13.1 Общие положения

**Редактор скриптов** предназначен для написания, редактирования и отладки скриптов в системе КРУГ-2000.

**Основные функции**

- Обработка текста: копирование, вставка, поиск, замена, удаление и другие текстовые операции с кодом скрипта
- Автоматизация создания текста скрипта: вставка в скрипт управляющих операторов, функций VBScript, встроенных функций, констант и других элементов
- Подсветка синтаксиса и контекстная справка
- Отображение информации об ошибке в скрипте, подсветка строки с ошибкой
- Работа с файлами: отображение дерева файлов скриптов с возможностью множественного выделения, копирования, удаления и переименования файлов; создания папок, файлов скриптов и библиотек
- Поиск текста в файлах скриптов и библиотек
- Отображение информации о работе скрипта в Генераторе динамики и лог-файла

## 13.2 Главное окно




Главное окно **Редактора скриптов** содержит:

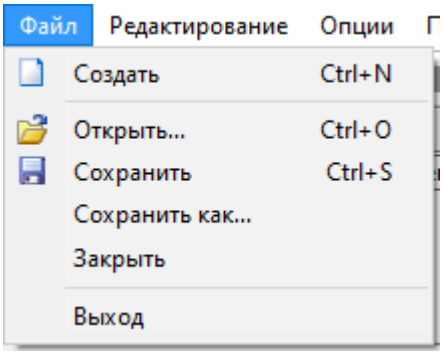
- Главное меню
- Панель инструментов
- Окно **Менеджер скриптов**
- Окно **Готовые конструкции**
- Рабочая область (область набора и редактирования текста скрипта)
- Информационное окно (внизу)
- Строка состояния.

## 13.3 Главное меню

Главное меню активизируется щелчком левой клавиши мыши или нажатием клавиши **ALT**. Пункты меню активизируются щелчком левой клавиши мыши или комбинацией горячих клавиш (например: для меню **Файл** – **ALT+Ф**).

### 13.3.1 Файл

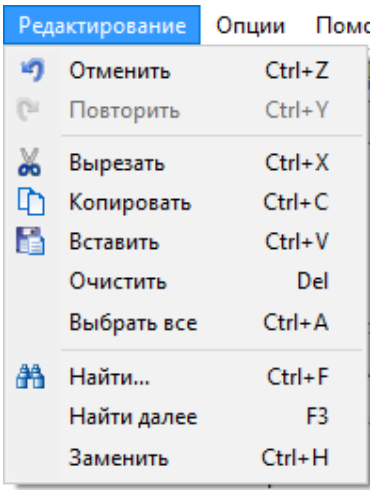
Структура меню **Файл:**




- **Создать** – создание нового файла скрипта
- **Открыть** – вызов стандартного диалога открытия файла с установленными по умолчанию расширениями ***.ks** (скрипты) и ***.kl** (библиотеки скриптов)
- **Сохранить** – сохранение открытого файла. Если файл еще ни разу не сохранялся на диске, выводится стандартный диалог сохранения файла, иначе файл сохраняется с именем, под которым он был открыт
- **Сохранить как…** – сохранение файла под новым именем
- **Закрыть** – закрытие открытого файла
- **Выход** – завершение работы Редактора скриптов.

### 13.3.2 Редактирование

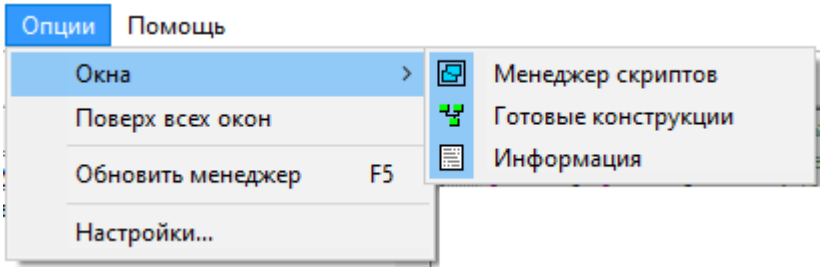
Структура меню **Редактирование:**




- **Отменить - Выбрать все** – функции обработки текста
- **Найти…** – поиск текста
- **Найти далее** – поиск найденного текста с текущей позиции
- **Заменить** – замена слов в тексте.

### 13.3.3 Опции

Структура меню **Опции:**



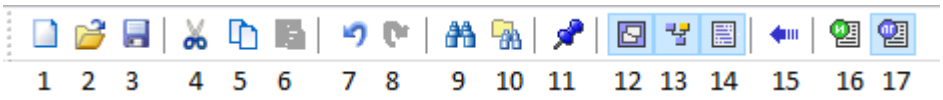

- **Окна\Менеджер скриптов** – открытие/закрытие окна **Менеджер скриптов**
- **Окна\Готовые конструкции** – открытие/закрытие окна **Готовые конструкции**
- **Окна\Информация»** - открытие/закрытие окна **Информация**
- **Поверх всех окон** – установка окна поверх других окон
- **Обновить менеджер скриптов** - обновление **Менеджера скриптов**
- **Настройки…** - вызов окна **Настройки**

### 13.3.4 Помощь

Структура меню **Помощь:**

- **Справка…** – вызов файла со справочной информацией.
- **О программе…** – коротко о Редакторе скриптов.

## 13.4 Панель инструментов




Кнопки панели:

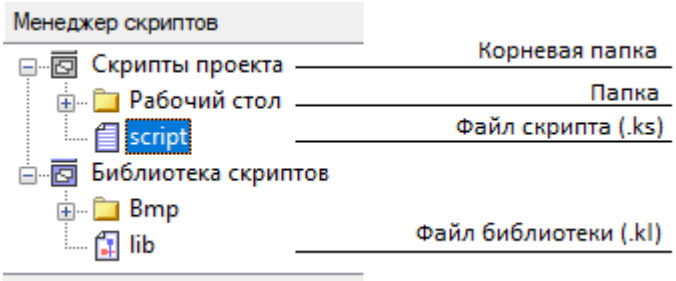
1 **Создать файл**
2 **Открыть файл**
3 **Сохранить файл**
4 **Вырезать**
5 **Копировать**
6 **Вставить**
7 **Отмена**
8 **Вернуть (отменить сделанные изменения)**
9 **Найти**
10 **Поиск в файлах**
11 **Поверх всех окон**
12 **Менеджер скриптов**
13 **Готовые конструкции**
14 **Информация**
15 **Поиск открытого файла в Менеджере скриптов**. Найденный файл подсвечивается. Если файл не найден – выводится сообщение
16 **Запретить/разрешить вывод информации** в окно **Информация времени выполнения**
17 **Запретить/разрешить вывод информации** в окно **Лог**.

Все кнопки снабжены всплывающими подсказками.

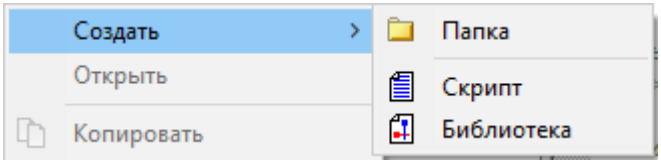
## 13.5 Менеджер скриптов

Окно **Менеджер скриптов** отображает древовидную структуру библиотек и файлов скриптов проекта.

Корневые папки не могут быть переименованы, скопированы или удалены. Их содержимым являются: другие папки, файлы скриптов или файлы библиотек скриптов.



Содержимое корневых папок отражает реальную структуру этих папок на диске. Например, если путь к скриптам проекта задается как **C:\Project\Scripts**, то путь к файлу **new3.ks** будет **C:\Project\Scripts\new3.ks**

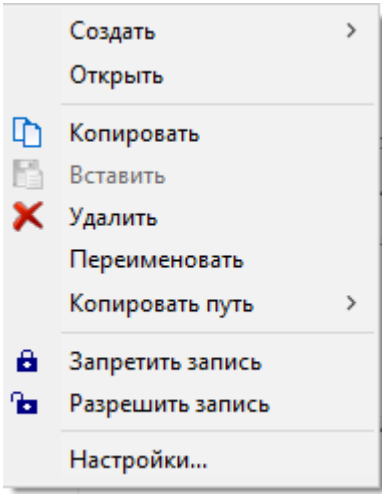



Каждая папка может включать в себя другие папки, файлы скриптов и файлы библиотек скриптов.

Файлы с расширениями отличными от **.ks** и **.kl** в Менеджере скриптов не отображаются.

Пути к папкам задаются в окне **Настройки**.

### 13.5.1 Действия над папками и файлами



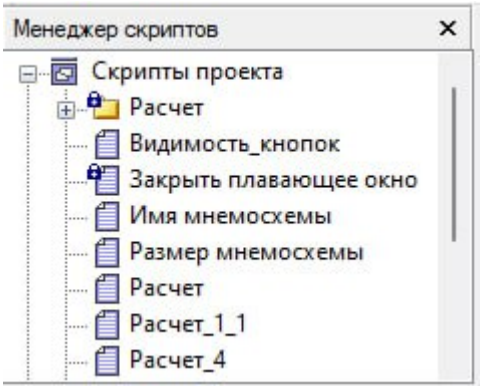

Менеджер скриптов поддерживает множественное выделение (при нажатой клавише **Shift** или **Ctrl**) и операции **Drag & Drop** внутри дерева скриптов.

Операции над папками и файлами осуществляются через контекстное меню.

Контекстное меню вызывается щелчком правой клавиши мышки на каком либо элементе дерева скриптов либо нажатием клавиш **Shif + F10**.

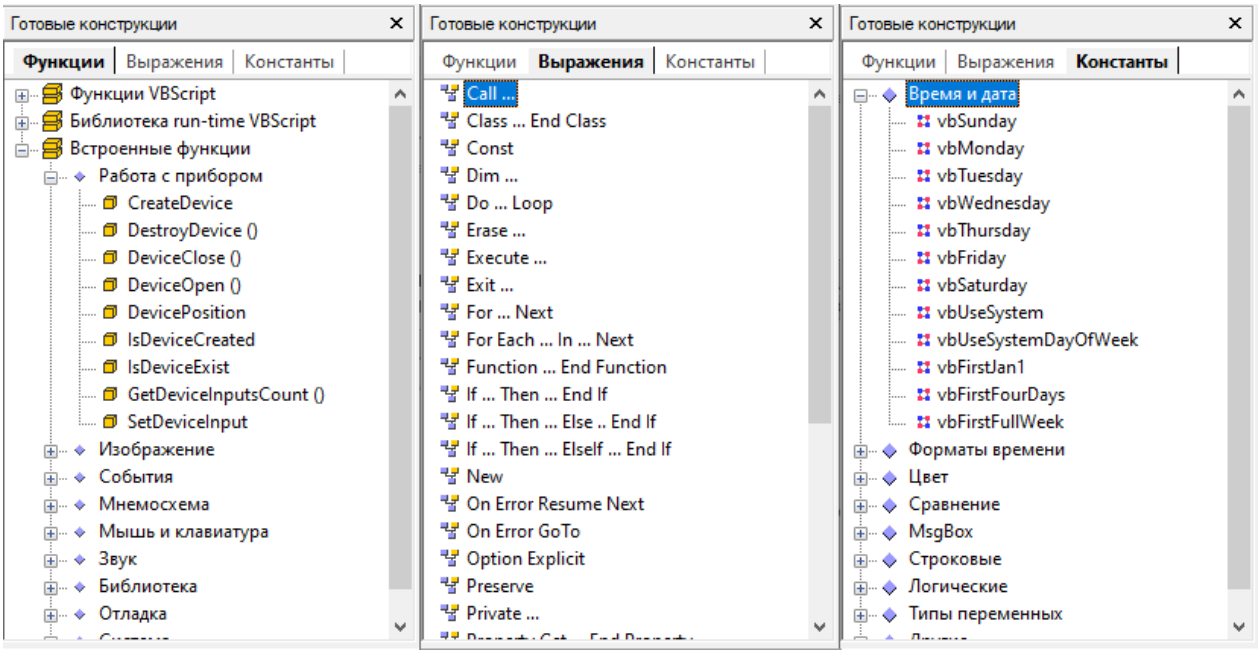
Операции в контекстном меню:

- **Создать\Папка** – вызов диалога создания папки
- **Создать\Скрипт** – вызов диалога создания файла скрипта
- **Создать\Библиотека** – вызов диалога создания библиотеки скрипта
- **Открыть** – открытие файл в редакторе
- **Копировать** – копирование выделенных элементов
- **Вставить** – вставка текста из буфера обмена
- **Удалить** – удаление выделенных элементов
- **Переименовать** – редактирование имени файла (папки)
- **Копировать путь\Абсолютный** – копирование абсолютного пути к файлу (папке)
- **Копировать путь\Относительный** – копирование относительного пути к файлу (папке). Относительный путь задается относительно корневой папки
- **Запретить запись** – запрещение перезаписи выделенных элементов. Такие файлы (папки) отображаются в виде иконки с замком.
- **Разрешить запись** – разрешение перезаписи выделенных элементов.
- **Настройки…** – вызов окна **Настройки**.



## 13.6 Готовые конструкции

Окно **Готовые конструкции** позволяет автоматизировать набор текста скрипта за счет вставки в текст готовых конструкций (функций, выражений и операторов, констант).




### 13.6.1 Функции

Закладка **Функции** разделяется на три ветки: **Функции VBScript**, **Библиотека run-time VBScript** и **Встроенные функции**.

Каждая ветка содержит сгруппированные по группам элементы и обозначается значком , за которым следует название группы.

Каждый элемент поддерживает следующие операции (это касается также элементов для закладок **Выражения** и **Константы**):

- Копирование
- Вызов справки по данному элементу
- Операции **Drag & Drop**.

### 13.6.2 Выражения

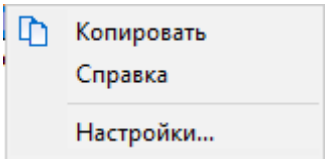
Элементы закладки **Выражения** поддерживают все операции для элементов закладки **Функции**.

### 13.6.3 Константы

Ветки закладки **Константы** содержат элементы, которые являются предопределенными константами VBScript.

### 13.6.4 Контекстное меню

Контекстное меню окна **Готовые конструкции** вызывается для какого-либо элемента в одной из закладок при нажатии правой клавиши мыши на элементе дерева либо при нажатии комбинации клавиш **Shift + F10**. Контекстное меню позволяет:



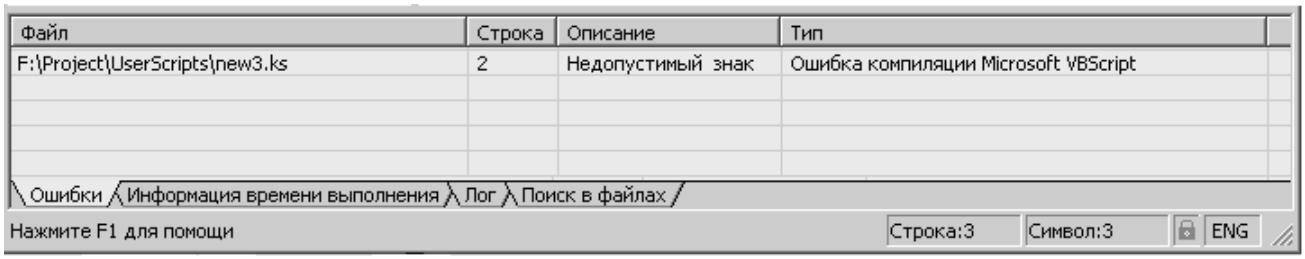
- **Копировать** – копировать элемент в буфер обмена
- **Справка** – вызвать справку по данному элементу
- **Настройки…** – вызвать диалог Настройки.

## 13.7 Информация

Окно **Информация** предназначено для вывода информации об ошибках скрипта, информации времени выполнения, лог файла и информации, найденной при поиске в файлах.

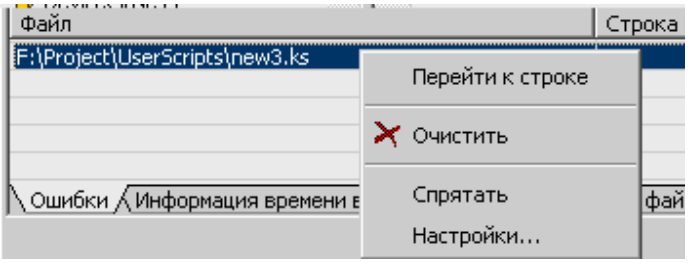
### 13.7.1 Закладка «Ошибки»

В процессе выполнения скрипта в Генераторе динамики в закладке **Ошибки** отображается информация о возникших ошибках в скрипте.




При двойном щелчке мышкой на строке с ошибкой в окне редактора открывается файл скрипта с ошибкой и «подсвечивается» строка с ошибкой.

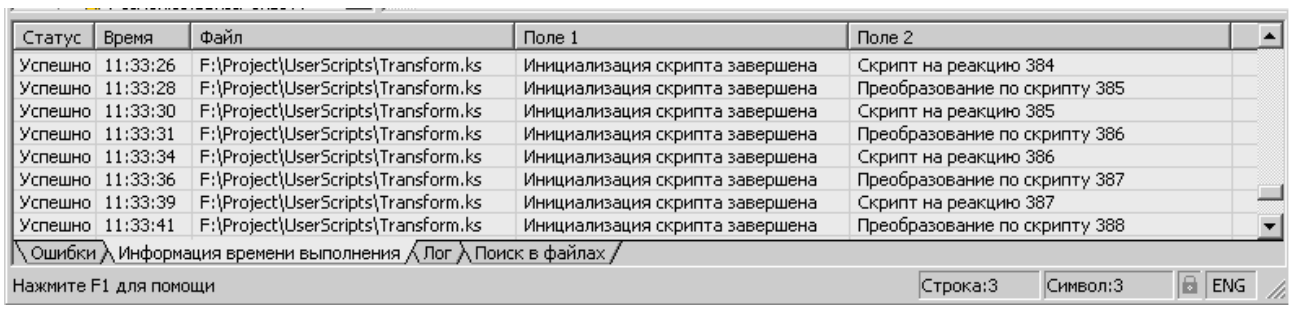
При нажатии на правую клавишу мыши либо при нажатии комбинации клавиш **Shift+F10** отображается контекстное меню:




- **Перейти к строке** – открытие файла с ошибкой и «подсвечивание» строки с ошибкой
- **Очистить** – удаление всего содержимого закладки **Ошибки**
- **Спрятать** – закрытие окна **Информация**
- **Настройки…** – вызов диалога Настройки

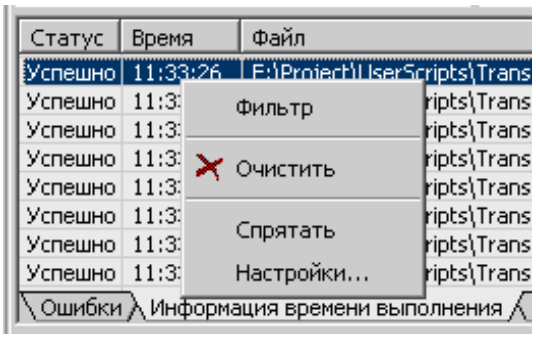
### 13.7.2 Закладка «Информация времени выполнения»

В процессе выполнения скрипта в Генераторе динамики в закладке **Информация времени выполнения** отображается информация о работе скрипта.



При нажатии на правую клавишу мыши или при нажатии комбинации клавиш **Shift+F10** отображается контекстное меню:

- **Фильтр** – вызов диалога **Фильтр**
- **Очистить** – очистка содержимого закладки **Информация времени выполнения**
- **Спрятать** – закрытие окна **Информация**
- **«Настройки…** – вызов диалога **Настройки**.

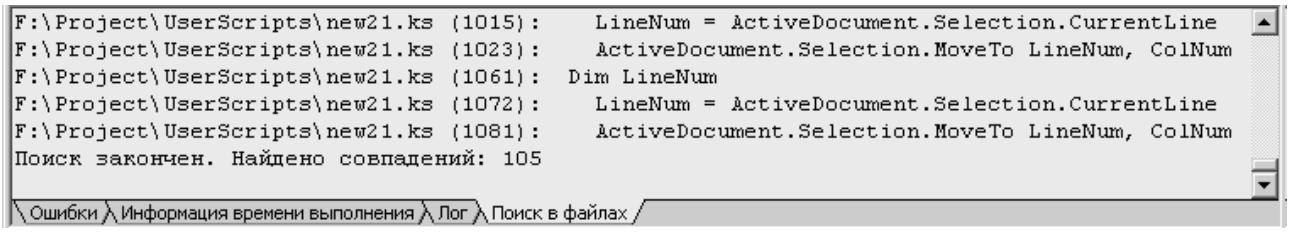



### 13.7.3 Закладка «Лог»

В закладке **Лог** отображается информация, записываемая в лог-файл.

### 13.7.4 Закладка «Поиск в файлах»

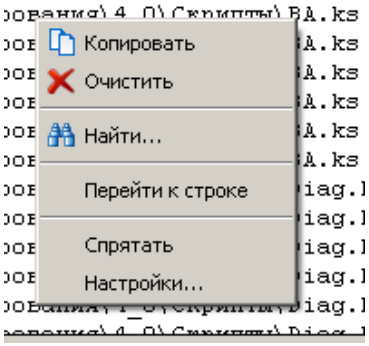
В закладке **Поиск** в файлах отображается информация о найденных в файлах текстах: словах, предложениях и т.п.




При двойном щелчке левой клавишей мыши на строке закладки **Поиск в файлах** открывается файл и «подсвечивается» найденная строка.

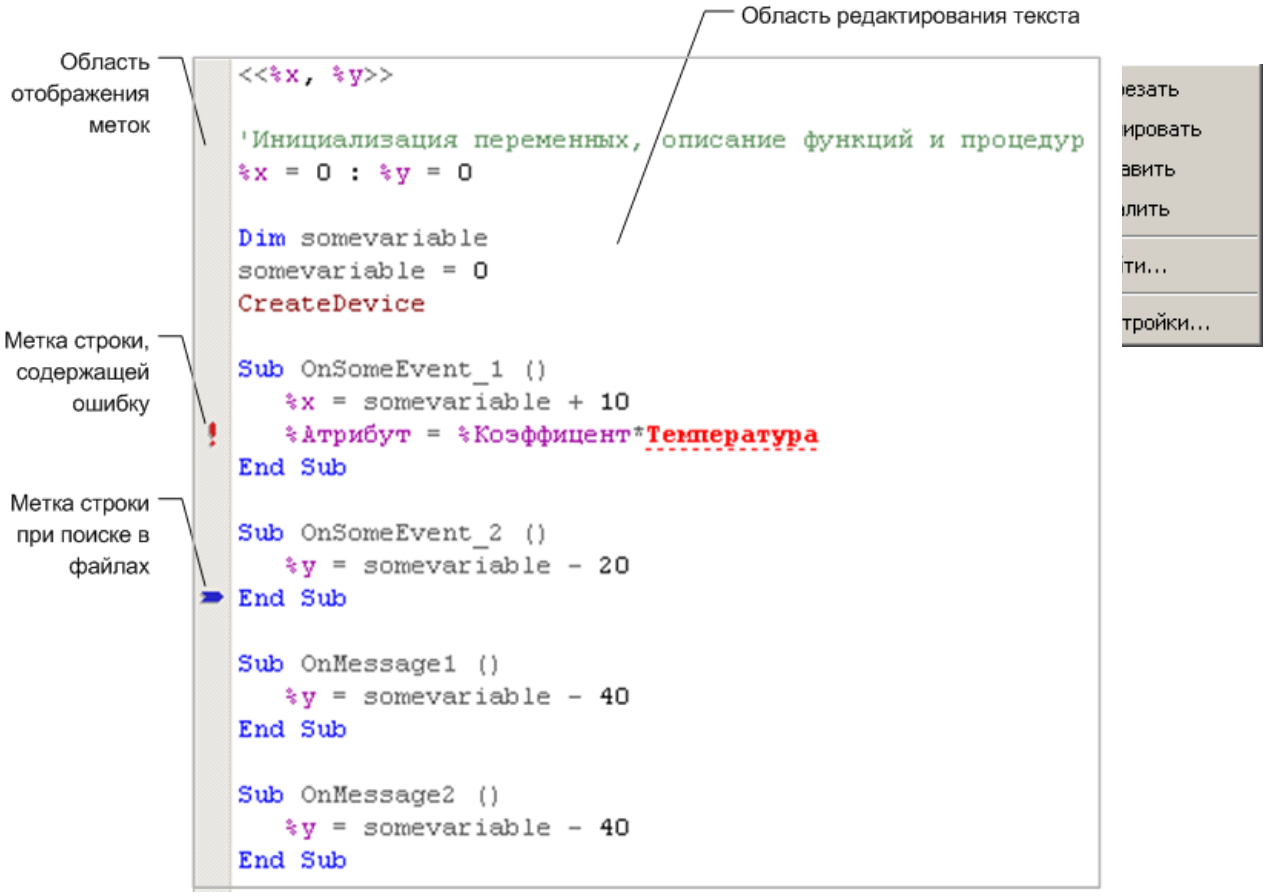
Контекстное меню окна содержит:

- **Копировать** – копирование выделенного фрагмента в буфер
- **Очистить** – очистка содержимого закладки **Поиск в файлах**
- **Найти** – вызов диалога поиска в тексте
- **Перейти к строке** – переход к строке с найденным текстом
- **Спрятать** – закрытие окна **Информация**
- **«Настройки…** – вызов диалога **Настройки**.




## 13.8 Область редактирования текста скрипта

Общий вид области редактирования текста скрипта показан на следующем рисунке.

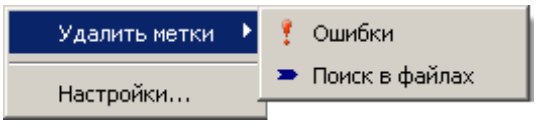



### 13.8.1 Контекстное меню в окне редактора

При нажатии на правую клавишу мыши или при нажатии комбинации клавиш **Shift+F10** в области редактирования на экране отображается контекстное меню.

- **Вырезать** – операция «вырезания» для выделенного фрагмента
- **Копировать** – копирование выделенного фрагмента в буфер
- **Вставить** – вставка текста из буфера обмена.
- **Удалить** – удаление выделенного фрагмента
- **Найти…** – вызов на экран диалога поиска в тексте
- **Настройки…** – вызов диалога **Настройки**.

При нажатии на правую клавишу мыши или при нажатии комбинации клавиш **Shift+F10** в области отображения меток на экране выводится контекстное меню.




- **Удалить метки\Ошибки** – удаление метки **Ошибка**
- **Удалить метки\Поиск в файлах** – удаление метку **«Поиск в файлах».**
- **Настройки…** – вызов диалог **Настройки.**

## 13.9 Диалоги

### 13.9.1 Диалог «Настройки»

Диалог **Настройки** включает:

- Общие настройки
- Настройки Менеджера скриптов
- Настройки готовых конструкций
- Настройки Редактора скриптов
- Настройки путей.

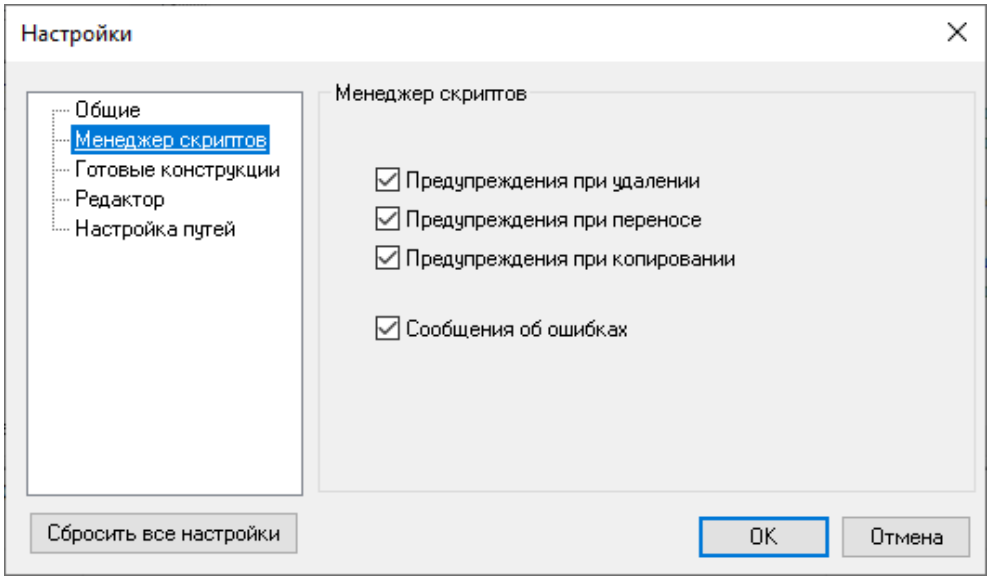
**ВНИМАНИЕ!!!**

**Кнопка «Сбросить все настройки» устанавливает настройки по умолчанию.**
**Все изменения в настройках вступают в силу только после нажатии на кнопку OK.**

**Общие настройки**

- **Сохранять файлы автоматически** – автоматическое сохранение файла при его закрытии (без вывода запроса на подтверждение)
- **Запрос перед сохранением** – вывод запроса на подтверждение сохранения файла перед закрытием файла
- **Полный путь к файлу в заголовке окна** –
- отображение в заголовке окна полного пути к файлу
- **Вставлять шаблон скрипта в новые файлы** – при создании нового файла скрипта или файла библиотеки скрипта в файл вставляется шаблон скрипта. Если данный признак не установлен – создается пустой файл.

**Настройки Менеджера скриптов**



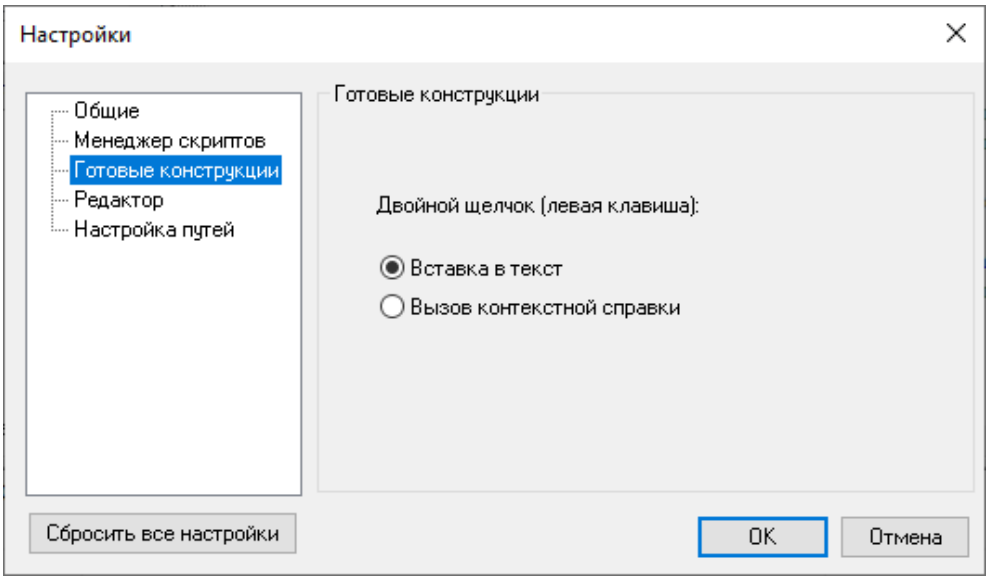

- **Предупреждения при удалении** – вывод запроса на подтверждение удаления при попытке удалить файл\папку из менеджера скриптов
- **Предупреждения при переносе** – вывод запроса на подтверждение операции **Drag & Drop** при переносе файла\папки в другую папку
- **Предупреждения при копировании** – вывод запроса на подтверждение операции **Drag & Drop** при копировании файла\папки в другую папку

- **Сообщения об ошибках** – вывод на экран сообщений при возникновении каких-либо ошибок при копировании, удалении и других операциях в Менеджере скриптов. Если признак не установлен – сообщение не выводится.

### Настройки готовых конструкций

В данной настройке определяется действие при двойном щелчке мыши на элементе в закладке **Готовые конструкции**:

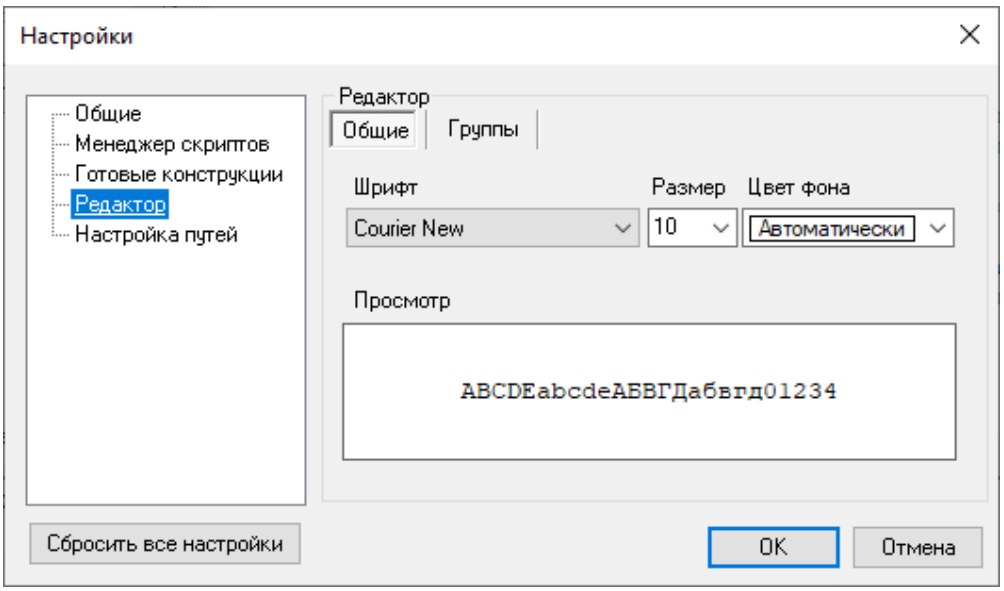
- **Вставка в текст** или **Вызов контекстной справки**.



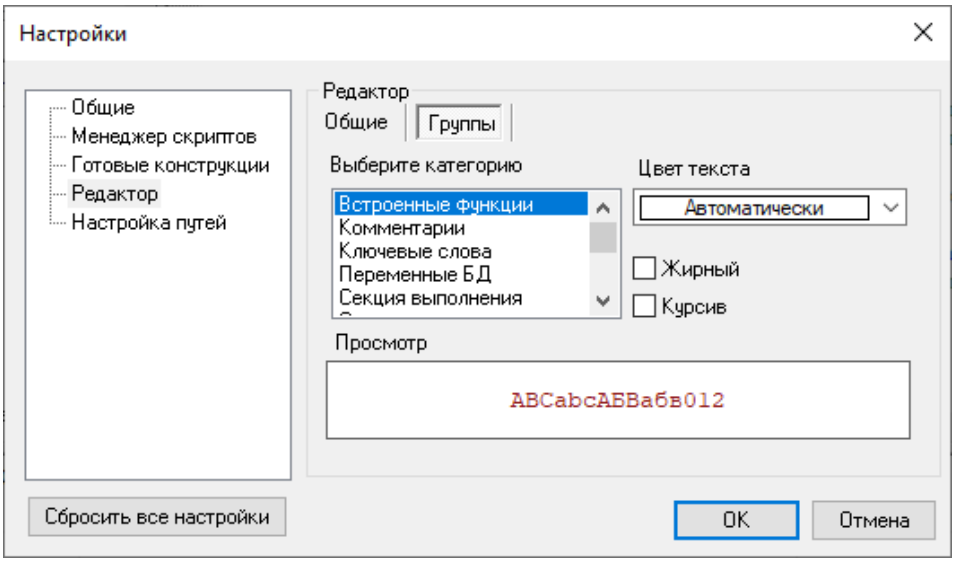

### Настройки Редактора скриптов

Данная настройка содержит закладки:

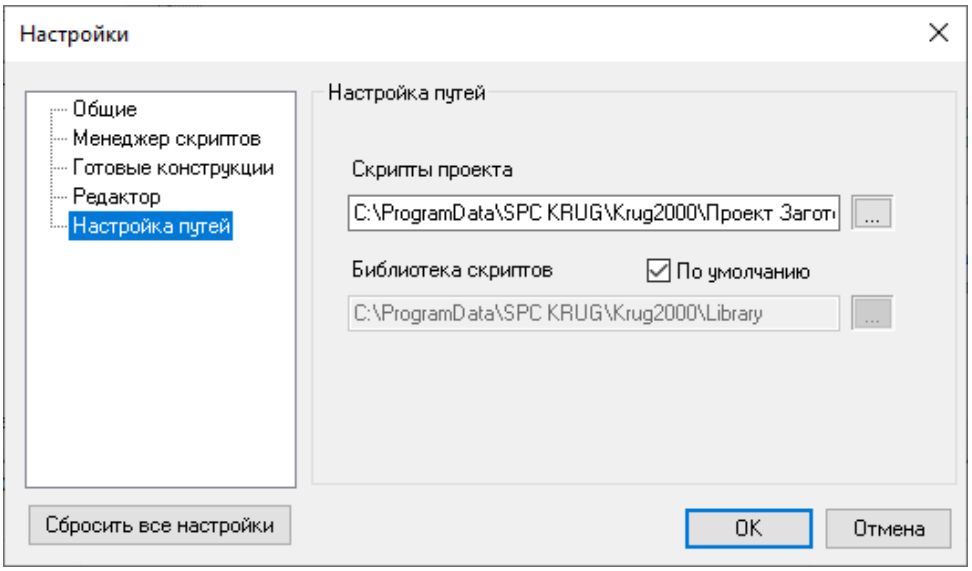
- **Общие.** Настраивается **Шрифт** и **Размер** текста, а также **Цвет фона**. В панели **Просмотр** отображается пример текста с заданными настройками




- **Группы.** Настройка категории текста скрипта – **Выберите категорию** – и задание параметров текста для выбранной категории: **Цвет текста, Жирный** и **Курсив**



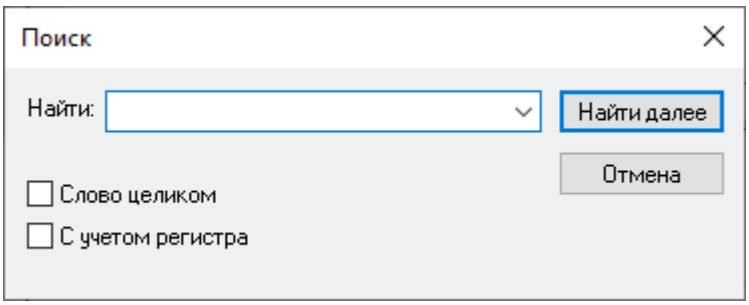

**Настройка путей**




- **Скрипты проекта** – установка пути к скриптам проекта в Менеджере скриптов
- **Библиотека скриптов** – установка пути к библиотеке скриптов в Менеджере скриптов
- **По умолчанию** – если данный признак установлен, то путь к библиотеке скриптов берется из переменной окружения **PathToLib** (в ней хранится путь к библиотеке системы КРУГ - 2000).

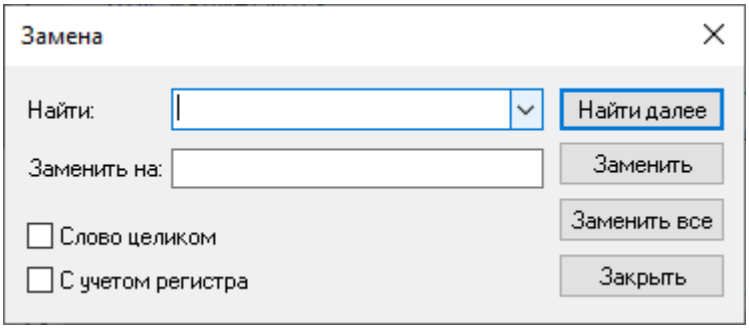
Для выбора пути используйте кнопку ....

### 13.9.2 Диалог «Поиск в тексте»



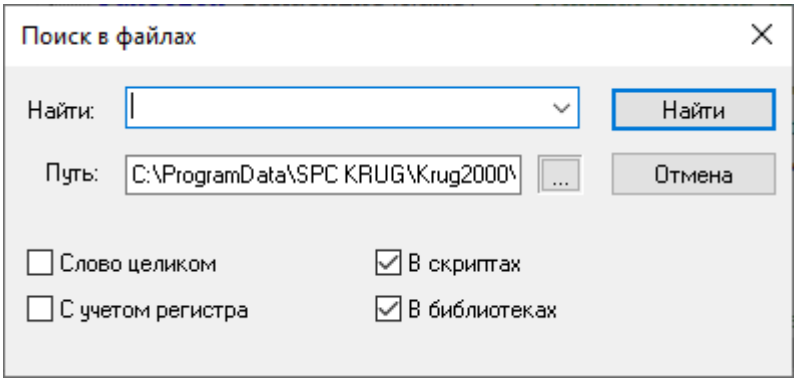

Диалог вызывается из меню, панели инструментов или при нажатии **Ctrl+F**.

### 13.9.3 Диалог «Замена в тексте»




Диалог вызывается из меню или при нажатии **Ctrl+H**.

### 13.9.4 Диалог «Поиск в файлах»




Поиск текста, заданного в поле **Найти**, осуществляется в файлах с расширением **.ks,** если отмечено **В скриптах**, или в файлах с расширением **.kl** – если отмечено **В библиотеках**

Поиск может осуществляться в соответствии с установленными признаками: **Слово целиком** и **С учетом регистра.**

## ПРИЛОЖЕНИЕ В. КОДЫ КЛАВИШ

КОДЫ КЛАВИШ

| ДЕСЯТИЧНОЕ ЗНАЧЕНИЕ | КЛАВИША КЛАВИАТУРЫ |
|---|---|
| 08 | BACKSPACE |
| 09 | TAB |
| 12 | CLEAR |
| 13 | ENTER |
| 16 | SHIFT |
| 17 | CTRL |
| 18 | ALT |
| 19 | PAUSE |
| 20 | CAPS LOCK |
| 27 | ESC |
| 32 | Пробел |
| 33 | PAGE UP |
| 34 | PAGE DOWN |
| 35 | END |
| 36 | HOME |
| 37 | Стрелка влево |
| 38 | Стрелка вверх |
| 39 | Стрелка вправо |
| 40 | Стрелка вниз |
| 44 | PRINT SCREEN |
| 45 | INS |
| 46 | DEL |
| 47 | HELP |

| ДЕСЯТИЧНОЕ ЗНАЧЕНИЕ | КЛАВИША КЛАВИАТУРЫ |
|---|---|
| 48 | 0 |
| 49 | 1 |
| 50 | 2 |
| 51 | 3 |
| 52 | 4 |
| 53 | 5 |
| 54 | 6 |
| 55 | 7 |
| 56 | 8 |
| 57 | 9 |
| 65 | A |
| 66 | B |
| 67 | C |
| 68 | D |
| 69 | E |
| 70 | F |
| 71 | G |
| 72 | H |
| 73 | I |
| 74 | J |
| 75 | K |
| 76 | L |
| 77 | M |

| ДЕСЯТИЧНОЕ ЗНАЧЕНИЕ | КЛАВИША КЛАВИАТУРЫ |
|---|---|
| 78 | N |
| 79 | O |
| 80 | P |
| 81 | Q |
| 82 | R |
| 83 | S |
| 84 | T |
| 85 | U |
| 86 | V |
| 87 | W |
| 88 | X |
| 89 | Y |
| 90 | Z |
| 96 | Numeric keypad 0 |
| 97 | Numeric keypad 1 |
| 98 | Numeric keypad 2 |
| 99 | Numeric keypad 3 |
| 100 | Numeric keypad 4 |
| 101 | Numeric keypad 5 |
| 102 | Numeric keypad 6 |
| 103 | Numeric keypad 7 |
| 104 | Numeric keypad 8 |
| 105 | Numeric keypad 9 |
| 106 | * |
| 107 | + |

| ДЕСЯТИЧНОЕ ЗНАЧЕНИЕ | КЛАВИША КЛАВИАТУРЫ |
|---|---|
| 109 | - |
| 111 | / |
| 112 | F1 |
| 113 | F2 |
| 114 | F3 |
| 115 | F4 |
| 116 | F5 |
| 117 | F6 |
| 118 | F7 |
| 119 | F8 |
| 120 | F9 |
| 121 | F10 |
| 122 | F11 |
| 123 | F12 |
| 144 | NUM LOCK |
| 145 | SCROLL LOCK |
| 160 | Левый SHIFT |
| 161 | Правый SHIFT |
| 162 | Левый CONTROL |
| 163 | Правый CONTROL |
| 164 | Левый MENU |
| 165 | Правый MENU |