OpusScript

Комментирование кода

Любой закомментированный текст в скрипте не обрабатывется как код. Комментирование предназначено для объяснения работы кода и других важных заметок, либо для выборочного отключения уже написанных строчек кода при тестировании программы (чтобы их не стирать и не набирать заново). Существует два типа комментирования: однострочное и многострочное. Несколько однострочных комментариев можно также использовать как многострочное комментирование, или многострочное комментирование можно использовать как однострочное если знаки объявления начала и конца комментария находятся в одной строчке. Скрипт, состоящий только из одних комментариев, вызовет сообщение об ошибке (а пустой скрипт нет). Перед финальной компиляцией публикации рекомендуется удалить все комментарии, для уменьшения размера исполняемого **EXE**-файла.

Однострочное комментирование.

[//] - знак объявления однострочного комментария.

Комментарием является весь текст строчки до её конца (абсолютно любой), стоящий после знака объявления комментария. Однострочное комментирование лучше подходит для коротких заметок в конце строчки написанного кода или для отключения (закомментирования) нужных строчек кода.

Многострочное комментирование.

[/*] - знак объявления начала многострочного комментария.

[*] - звездочка ставится в начале каждой строчки многострочного комментария (рекомендуется, но необязательно).

[*/] - знак объявления конца многострочного комментария.

Комментарием является весь текст после знака объявления начала комментария до знака объявления конца комментария. Пока не объявлен конец комментария, весь текст (даже без знака [*] в начале) на следующих строчках будет считаться комментарием. Если знаки начала и конца комментария расположены в одной строчке, то если поставить точку с запятой [;], можно продолжить писать код в этой строчке после комментария.

Пример:

```
Debug.trace("Жираф") //обычный однострочный комментарий
wait(0)
//многострочный комментарий
//с помощью двух однострочных
Debug.trace("\nНосорог") /*многострочный комментарий как однострочный*/
wait(0)
/*
обычный
многострочный
комментарий
*/
wait(0)/*комментарий внутри строчки кода*/;Debug.trace("\nТапир")
```

Функция

Object.Function(parameters)

Пояснение:

Object - подходящий объект, на который подействует функция. Если функция глобальная, то объект не указывается. Если одна функция возвращает объект, а другая функция может подействовать на этот объект, то вместо объекта можно указать функцию, возвращаемую объект:

Function(parameters).Function(parameters) ...

parameters - параметры функции (через запятую, если их несколько). Скобки указываются в любом случае, даже если у функции нет параметров.

Описание каждой функции поделено на разделы: синтаксис, возврат, параметры и пример.

Синтаксис:

Раздел, показывающий название функции и регистр, в котором она должна быть введена. Ввод названия функции должен быть произведен без пробелов между словами и скобками. При наборе скриптов программа автоматически исправляет регистр введенных функций, а рядом появляется краткая всплывающая подсказка. На одно только автоисправление полагаться не стоит, например **If** на **if** не исправляется. Также автоисправление неправильно действует на объекты **Number()** и **String()**, делая их с маленькой буквы (**number()** и **string()**), что вызывает ошибку.

Возврат:

Раздел, присутствующий в описании, когда функция возвращает значение:
(Ч) - числовое,
(Ц) - целочисленное,
(Б) - логическое (булево),
(С) - строковое,
(О) - объект.

Параметры:

Раздел, присутствующий в описании, когда у функции есть параметры. К каждому параметру функции дается пояснение и указывается какой тип значения требуется для параметра:
(Ч) - числовое,
(Ц) - целочисленное,
(Б) - логическое (булево),
(С) - строковое,
(О) - объект.
(Л) - любое.
При вызове функции вместо конкретного значения параметра, можно использовать:
1) переменную, содержащую требуемый тип значения;
2) другую функцию, возвращающую требуемый тип значения.
Если функция содержит несколько параметров, то они разделяются запятыми. Порядок ввода параметров должен соответствовать синтаксису функции, чьи значения следует вводить поочередно, через запятую и не меняя местами друг с другом. В описании указывается ввод каких параметров необязателен (**ОП** - обязательный параметр, **НП** - необязательный параметр). Необязательный параметр подразумевает значение по умолчанию, которое используется, если не вводить собственное значение. Необязательный параметр можно **не указывать** только если **указано значение** предыдущего необязательного параметра (если такой есть), а значения всех последующих необязательных параметров **также не указаны**.

Пример:

Раздел, демонстрирующий действие функции.

Функции скриптовой консоли Debug.trace и Debug.tracePane

Функции **Debug.trace** и **Debug.tracePane** выводят данные в скриптовую консоль и могут быть использованы для отладки программы. Например, можно отобразить значения переменных, чтобы проверить, соответствуют ли они ожидаемым значениям или нет. Отображение скриптовой консоли недоступно, когда публикация скомпилирована, поэтому эти функции не окажут отрицательного влияния на итоговую публикацию. Однако рекомендуется закомментировать или удалить любые отладочные инструкции перед компиляцией публикации. Вкладка **"Variable Watch"** скриптовой консоли позволяет выборочно отображать только значения системных переменных и переменных, объявленных пользователем в настройках публикации (пункт меню **"Edit"** или **"Publication"**, диалог **"Publication Properties"**, вкладка **"Variables"**). Функция **Debug.tracePane** является расширенной версией функции **Debug.trace** и отличается от неё возможностью выводить данные в скриптовую консоль не только во вкладке **"Default"**, но и в любой другой с названием, заданным пользователем.

Синтаксис:

```
Debug.trace(String)
Debug.tracePane(PaneName, String)
```

Параметры:

String - **(Л)**, значения, переменные и выражения для вывода. **ОП**.

PaneName - **(С)**, название панели в скриптовой консоли для вывода данных. Может быть одним из существующих **"Default"**, **"Errors"** или любым другим новым. **ОП**.

Примеры:

Вывод простых значений:

```
Debug.trace(10+10) //отобразить в консоли: 20
Debug.trace(10+""+10) //отобразить в консоли: 1010
Debug.trace("Количество зверей: "+100) //отобразить в консоли: Количество зверей: 100
```

Вывод значений переменных:

```
zootype="Чепрачный тапир"
zoosum=13
zoopark=true
Debug.trace(zootype+": "+zoosum+" "+zoopark) //отобразить в консоли: Чепрачный тапир: 13 true
```

Вывод значения выражения:

```
x=2;y=Math.pow(x, 2);z=3
Debug.trace(y*z) //отобразить в консоли: 12
```

Вывод значений на разных вкладках:

```
zootype1="Бизон"
zootype2="Слон"
zoosum1=9
zoosum2=5
Debug.tracePane("Animals", zootype1+"\n"+zootype2) //отобразить названия зверей в отдельной вкладке
Debug.tracePane("Numbers", zoosum1+"\n"+zoosum2) //отобразить числа в отдельной вкладке
```

Создание пользовательских функций

Пользовательская функция - это определяемая единожды часть кода, которую можно затем многократно использовать как обычную встроенную функцию. Функция определяется только в скриптовом объекте страницы, а не в скриптовом объекте дочернего элемента страницы или скрипте действия, иначе она не заработает. Код содержащийся в функции не выполнится, пока не произойдет вызов функции.

Синтаксис:

Создание (определение) пользовательской функции:

function functionName(parameters) {list_of_statements}

Пояснение:

function - ключевое слово, объявляющее функцию.

functionName - название функции, заданное пользователем.

parameters - параметры функции (через запятую), определяемые пользователем.

() - параметры должны быть заключены в круглые скобки.

list_of_statements - часть кода, которая будет выполняться при вызове функции.

{} - часть кода должна быть заключена в фигурные скобки.

Вызов пользовательской функции:

functionName(parameters)

Пояснение:

functionName - название уже определенной пользователем функции.

parameters - набор параметров (через запятую), передаваемых в функцию для обработки.

() - параметры должны быть заключены в круглые скобки.

Примеры:

Создание функции в скриптовом объекте страницы.

```
function CustomFn1(VectorName, Amount) //определить функцию CustomFn1 с двумя параметрами
{
VectorName.Move(Amount, Amount, Amount*0.02, false)
VectorName.Spin(180, Amount*0.01, true)
VectorName.Roll(180, Amount*0.01, true)
}
```

Вызов функции из скриптового объекта страницы.

```
CustomFn1(Vector1, 500) //выполнить функцию CustomFn1 с указанными значениями параметров
```

Вызов функции из скриптового объекта дочернего элемента страницы, на котором применяется функция.

```
CustomFn1(this, 500) //выполнить функцию CustomFn1 с указанными значениями параметров
```

Определение модифицированной функции без параметров.

```
function CustomFn2() //определить функцию CustomFn2 без параметров
{
VectorName.Move(Amount, Amount, Amount*0.02, false)
VectorName.Spin(180, Amount*0.01, true)
VectorName.Roll(180, Amount*0.01, true)
}
```

Вызов модифицированной функции без параметров из любого скриптового объекта.

```
VectorName=Vector1;Amount=500;CustomFn2()
```

Ключевое слово this

Ссылка на объект без указания его названия, полезно для универсализации скриптов. Обозначает **объект-хозяин** скриптового объекта, из которого происходил **вызов** функции, ссылающейся на неопределённый объект с помощью **this**.

Ключевое слово fork

Выполнить несколько функций не по очереди, а **одновременно**, если все эти функции начинаются с **fork** и написаны в скрипте друг за другом.

Синтаксис:

fork(Function, ObjectName)

Параметры:

Function - название выполняемой функции (должна не содержать параметров). **ОП**.

ObjectName - **(O)**, на который подействует функция (нельзя использовать **this**). **НП**, если вызываемые функции действуют на конкретные уже указанные в них объекты, иначе **ОП**.

Примеры:

Первая пользовательская функция без параметров является контейнером функций с параметрами, но с неопределенными объектами. Вызов второй пользовательской функции приводит к вызову первой сразу для трёх указанных объектов одновременно (this из первой функции будет ссылкой не на объект из которого происходил её вызов, а на объект указанный в функции fork).

```
function Twist() //определить функцию Twist без параметров
{this.Spin(180, 3, false)
this.Roll(180, 3, true)}
function Triplet() //определить функцию Triplet без параметров
{fork(Twist, Vector1);fork(Twist, Vector2);fork(Twist, Vector3)}
wait(1);Triplet()
```

Определены три пользовательских функции без параметров, являющихся контейнерами функций с параметрами и указанными объектами. Вызов четвертой пользовательской функции приводит к одновременному действию первых трех.

```
function Animation1() {Vector1.Spin(180, 3, false);Vector1.Roll(180, 3, true)}
function Animation2() {Vector2.Spin(180, 3, false);Vector2.Roll(180, 3, true)}
function Animation3() {Vector3.Spin(180, 3, false);Vector3.Roll(180, 3, true)}
function TripleAnima() {fork(Animation1);fork(Animation2);fork(Animation3)}
wait(1); TripleAnima()
```

Функция eval

Выполнить содержимое строкового значения как фрагмент кода. Если содержимое кодом не является, то выполнение становится невозможным и произойдет ошибка в программе. Данная функция полезна для выполнения кода из внешних текстовых файлов, когда программа уже скомпилирована, но необходимо модифицировать код.

Синтаксис:

eval(string)

Параметры:

string - **(C)**, фрагмент кода по содержанию. **ОП**.

Пример:

```
//Результатом будет вывод фразы "Визит к минотавру" в скриптовой консоли
a="Визит к " //переменной a присвоено строковое значение
b="Debug." //переменной b присвоено строковое значение
c="trace" //переменной c присвоено строковое значение
d='(a+"мино")' //переменной d присвоено строковое значение, а не выражение
eval (b+c+d) //выполняется как код текст, сложенный из строковых значений трех переменных
eval ('Debug.trace("тавру")') //строковое значение выполняется как код
```

Типы данных

Скрипт поддерживает шесть типов данных: число, логическое (булево) значение, строка, объект, нулевой, неопределенный. Переменная может содержать любой тип данных из вышеперечисленных.

Числовые значения:

Числовые значения обрабатываются в десятичной системе счисления, могут быть целыми или дробными (числами с плавающей точкой). Перед числовыми значениями долей единицы символ **0** необязателен. Окончание **eN** (где **N** - целое положительное или отрицательное число) в десятичном числовом значении используется для умножения числа на **10** в **N** степени.

Целые числа, идущие после символа **0** (должны состоять из цифр от **0** до **7**, иначе выведется ошибка), воспринимаются как числа в **восьмеричной системе счисления** и сразу переводятся в **десятичную**.

Целые числа, идущие после символов **0x** (должны состоять из цифр от **0** до **9** и букв от **A** до **F**, иначе выведется ошибка), воспринимаются как числа в **шестнадцатеричной системе счисления** и сразу переводятся в **десятичную**.

Примеры:

Десятичное целое число.

```
Debug.trace(32987) //результат: 32987
```

Восьмеричное целое число.

```
Debug.trace(0764) //результат: 500
```

Шестнадцатеричное целое число.

```
Debug.trace(0xFFFF) //результат: 65535
```

Числа с плавающей точкой.

```
Debug.trace(3.1415+" "+0.25+" "+.001) //результат: 3.1415 0.25 0.001
```

Сложение чисел введенных различными способами.

```
Debug.trace(26e2+11+0.11+011+0x11+48e-4) //2600+11+0.11+9+17+0.0048=2637.1148
```

Деление числовых значений на 0.

```
x=1 //сюда можно подставить любое положительное число в любой системе счисления
Debug.trace(x/0) //результат: 1.#INF
Debug.trace("\n"+(x/0).toString()+"\n") //результат: Infinity
Debug.trace(-x/0) //результат: -1.#INF
Debug.trace("\n"+(-x/0).toString()+"\n") //результат: -Infinity
Debug.trace(-x/0+x/0) //результат: -1.#IND
Debug.trace("\n"+(-x/0+x/0).toString()) //результат: NaN
```

Строковые значения:

Строковое значение - это любая текстовая комбинация из букв и цифр, заключенная в одинарные или двойные кавычки (тип кавычек должен быть одинаковым с каждой стороны). Строковое значение может содержать в себе символы кавычек, только если их тип отличается от того, в которые строковое значение заключено.

```
name="Rumburak";version='Opus Pro 9.75';book='~"Arabian Nights"~'
Debug.trace(name+"\n"+version+"\n"+ book)
```

Операции со строковыми значениями.

Строковые значения объединяются с помощью знака плюс.

```
Part1="Arabian ";Part2="Nights";Debug.trace(Phrase=Part1+Part2)
```

При сложении строкового значения с числовым возвращается строковое значение.

```
Word1="Room ";Number1=1408;Debug.trace(Word1+Number1) //результат: "Room 1408"
```

Когда числа заключены в кавычки, они считаются строковым, а не числовым значением. При сложении строковых значений из цифр между собой или с числами возвращается строковое значение.

```
Debug.trace("12"+"34"+56+78) //результат: строковое значение "12345678"
```

При вычитании, умножении и делении строковые значения, содержание которых представляет из себя число в десятичной системе счисления (целые и дробные, отрицательные и положительные), преобразуются в числовые значения. Все лишние символы **0** впереди отбрасываются.

```
Debug.trace("0036.5"-"12.5"+"\n") //результат: числовое значение 24
Debug.trace("3"*4*"0003"+"\n") //результат: числовое значение 36
Debug.trace("36"/4/"0003"+"\n") //результат: числовое значение 3
```

При вычитании, умножении и делении с участием строковых значений, содержание которых **НЕ** представляет из себя число в десятичной системе счисления, возвращается значение **NaN** (**Not-A-Number**).

```
Debug.trace("Word"-"ord") //результат: -1.#IND
Debug.trace("\n"+("Word"-"ord").toString()) //результат: NaN
```

Специальные символы в строковом значении.
Символ обратного слеша [\ ] зарезервирован для вывода специальных символов в строковом значении. Одинарный обратный слеш внутри строкового значения без специального символа справа не учитывается.
Debug.trace("И\т\о\г="+("ABC"==="A\B\C")) //результат: "Итог=true"
\xHH - вывести **ASCII**-символ по его коду, где **HH** - двузначное число в шестнадцатиричной системе счисления.
\uHHHH - вывести **Unicode**-символ по его коду, где **HHHH**-четырехзначное число в шестнадцатиричной системе счисления.
**** - использовать символ обратного слеша в строковом значении (**\x5c**). Чаще всего применяется для указания файлового пути.
Debug.trace("c:\\Games\\Soulbringer\\Readme.txt")
\' - использовать символ одинарных кавычек в строковом значении (**\x27**).
\" - использовать символ двойных кавычек в строковом значении (**\x22**).
\n - **Line Feed**, переход на новую строчку (**\x0a**).
Debug.trace("1-я строчка\n2-я строчка\n3-я строчка")
\r - **Carriage Return**, также переход на новую строчку (**\x0d**).
Debug.trace("1-я строчка\r2-я строчка\r3-я строчка")
\t - **Horizontal Tab**, горизонтальная табуляция (**\x09**).
Debug.trace("Column#1\tColumn#2\tColumn#3")
\0 - **Terminal Null**, окончание строкового значения (**\x00**).
Debug.trace("Абра\0Кадабра") //в скриптовой консоли выводится только "Абра"
\b - **Backspace**, нет практического использования (**\x08**).
\f - **Form Feed**, нет практического использования (**\x0c**).

Булевы значения:
Существует только два булевых значения: **true** (истина) и **false** (ложь). Эти значения эквивалентны словам **ДА** и **НЕТ** и являются основой для принятия решений в скрипте.

Объекты:
Переменная может также содержать значения нового объекта.

Нулевой:
Нулевой тип данных имеет только одно значение - **null**. Он используется для сложных скриптов, обычно для указания отсутствия какого-либо конкретного значения.

Неопределенный:
Неопределенный тип данных имеет только одно значение - **undefined**. Любая переменная или свойство, которые не были инициализированы, имеют неопределенный тип.

Функции преобразования данных

Список функций:

isFinite - проверить данные на конечность и бесконечность.
typeof - получить тип данных.
void - получить неопределенное значение.
valueOf - преобразовать данные в примитивное значение.
toString - преобразовать данные в строковое значение.
RGB - преобразовать количества красного, зеленого и синего в значение цвета.
parseInt - преобразовать данные в целое число.
parseFloat - преобразовать данные в число с плавающей точкой.

isFinite

Проверить является ли значение конечным числом.

Синтаксис:

isFinite(Value)

Возврат:

(Б), конечность значения (**true** - значение является конечным числом, **false** - значение бесконечно или не число).

Параметры:

Value - **(Л)**, проверяемое значение.

Пример:

```
Debug.trace("Булево значение: "+isFinite(false)+"\n") //true
Debug.trace("Строковое значение 999: "+isFinite("999")+"\n") //true
Debug.trace("Числовое значение 999: "+isFinite(999)+"\n") //true
Debug.trace("Числовое значение 0xFFFF: "+isFinite(0xFFFF)+"\n") //true
Debug.trace("Строковое значение 0xFFFF: "+isFinite("0xFFFF")+"\n") //false
Debug.trace("Числовое значение 0: "+isFinite(0)+"\n") //true
Debug.trace("Число стремящееся к 0: "+isFinite(Number.MIN_VALUE)+"\n") //true
Debug.trace("Максимально возможное число: "+isFinite(Number.MAX_VALUE)+"\n") //true
Debug.trace("10 в степени 308: "+isFinite(Math.pow(10, 308))+"\n") //true
Debug.trace("10 в степени 309: "+isFinite(Math.pow(10, 309))+"\n") //false
Debug.trace("Not-a-Number: "+isFinite(Number.NaN)+"\n") //false
Debug.trace("Отрицательная бесконечность: "+isFinite(Number.NEGATIVE_INFINITY)+"\n") //false
Debug.trace("Положительная бесконечность: "+isFinite(Number.POSITIVE_INFINITY)+"\n") //false
```

typeof

Получить тип данных.

Синтаксис:

typeof(Any)

Параметры:

Any - **(Л)**, проверяемые данные.

Возврат:

(С), название типа данных.

Пример:

```
Debug.trace(typeof(false)+"\n") //результат: boolean
Debug.trace(typeof("ABC")+"\n") //результат: string
Debug.trace(typeof(100)+"\n") //результат: number
Debug.trace(typeof(function () {})+"\n") //результат: function
Debug.trace(typeof(new Object())+"\n") //результат: object
Debug.trace(typeof(null)+"\n") //результат: object
Debug.trace(typeof(undefined)+"\n") //результат: undefined
```

void

Получить неопределенное значение.

Синтаксис:

void(Expression)

Возврат:

undefined

Параметры:

Expression - **(Л)**. **ОП**.

Пример:

```
A="One", B="Two";C=void(C=A, A=B, B=C) //сначала выполнится выражение в скобках
Debug.trace("a="+A+"\n"+"b="+B+"\n"+"c="+C) //результат: a=Two b=One c=undefined
```

valueOf

Преобразовать любое значение в примитивное. Полезно для объектов **Number**, **String**, **Boolean**, **Date**. Если сравнить одинаковые значения двух таких объектов одного типа с помощью строгого равенства, эти значения не будут равны, а если преобразовать эти значения в примитивные, то они станут равны.

Синтаксис:

Value.valueOf()

Возврат:

Примитивное значение. Для объекта **Date** значение будет в миллисекундах.

Параметры:

Value - **(Л)**, значение для преобразования. Буквальное значение (или выражение) должно быть заключено в круглые скобки.

Пример:

```
Numb1=new Number(8) //создание нового числового объекта со значением 8
Numb2=new Number(8) //создание второго числового объекта с таким же значением
Word1=new String("Z") //создание нового строкового объекта со значением "Z"
Word2=new String("Z") //создание второго числового объекта с таким же значением
if (Numb1===Numb2 || Word1===Word2) //строгое сравнение объектов с одинаковыми значениями
{Debug.trace("Обезьяна")} //сообщение не отобразится в скриптовой консоли
PrimNumb1=Numb1.valueOf() //преобразование в примитив
PrimNumb2=Numb2.valueOf() //преобразование в примитив
PrimWord1=Word1.valueOf() //преобразование в примитив
PrimWord2=Word2.valueOf() //преобразование в примитив
if (PrimNumb1===PrimNumb2 && PrimWord1===PrimWord2) //строгое сравнение
{Debug.trace("Лошадь")} //теперь сообщение отобразится в скриптовой консоли
```

toString

Преобразовать любое значение в строковое. Также действует с объектами (например **Math**, **Number**, **String**, **Boolean**, **Date**, **File**), делая их значения строковым или преобразуя название типа объекта в строковое значение.

Синтаксис:

Value.toString(NumeralSystem)

Возврат:

(С).

Параметры:

Value - **(Л)**, значение для преобразования. Буквальное значение (или выражение) должно быть заключено в круглые скобки.

NumeralSystem - **(Ц)** от **2** до **36**, система счисления. **НП**, по умолчанию **10**.

Пример:

```
Debug.trace((10+2*8).toString(16)+"\n") //выражение преобразуется в строку
Bool1=new Boolean(1);Bool2=Bool1.toString() //булевый объект преобразуется в строку
Numb1=new Number(39);Numb2=Numb1.toString() //числовой объект преобразуется в строку
Pi1=Math.PI;Pi2=Math.PI.toString() //математический объект преобразуется в строку
Debug.trace("Как число: "+(Bool1+Numb1+Pi1)+"\n")
Debug.trace("Как строка: "+(Bool2+Numb2+Pi2)+"\n")
Obj1=Vector1.GetChild(0).toString() //преобразование названия полигона в строку
Debug.trace("Название объекта: "+Obj1)
```

RGB

Преобразовать числа в значение цвета, которое используется с функциями графических объектов.

Синтаксис:

RGB(Red, Green, Blue)

Возврат:

(Ц), цвет, соответствующий полученому из трех значений: количества красного цвета, зеленого и синего.

Параметры:

Red - **(Ц)** от **0** до **255**, количество красного в цвете. **НП**, по умолчанию **0** (отсутствие красного).
Green - **(Ц)** от **0** до **255**, количество зеленого в цвете. **НП**, по умолчанию **0** (отсутствие зеленого).
Blue - **(Ц)** от **0** до **255**, количество синего в цвете. **НП**, по умолчанию **0** (отсутствие синего).

Пример:

```
var Fire=52, Grass=13, Water=113
var Color1=RGB(Fire, Grass, Water) //фиолетовый цвет
Debug.trace("Цвет: "+Color1.toString(16))
```

parseInt

Преобразовать любое значение в целое число. Также действует с объектами (например **Math**, **Number**, **String**, **Boolean**, **File**), делая их значения целым числом. Для объекта даты и времени используется специальная функция **Date.parse**.

Синтаксис:

parseInt(Value, NumeralSystem)

Возврат:

(Ц), в десятичной системе счисления.

Параметры:

Value - **(Л)**, для преобразования. **ОП**.
NumeralSystem - **(Ц)** от **2** до **36**, система счисления из которой преобразуется значение. **НП**, по умолчанию **10**.

Пример:

```
Debug.trace(parseInt("0764", 10)+"\n") //764
Debug.trace(parseInt("0764", 8)+"\n") //500
//при преобразовании строкового значения, начинающегося с 0 без указания системы счисления,
//цифра следующая за 0 игнорируется, а последующие символы пребразуются
//из шестнадцатиричной системы счисления в десятичную, поэтому ниже
Debug.trace(parseInt("0764")+"\n") //100, равносильно parseInt("0x64", 16)
Debug.trace(parseInt("Z9", 36)+"\n") //1269
Debug.trace(parseInt("ABC", 16)+"\n") //2748
Debug.trace(parseInt("0xABC", 16)+"\n") //2748
//число шестнадцатиричной формы само преобразуется в десятичное, поэтому ниже
Debug.trace(parseInt(0xABC, 16)+"\n") //шестнадцатиричное 2748 преобразуется в десятичное 10056
Debug.trace(parseInt(Math.PI)) //3
Debug.trace(parseInt("8.5")+"\n") //8
Debug.trace(parseInt("13 14 15")+"\n") //13
Debug.trace(parseInt("40 штук")+"\n") //40
Debug.trace(parseInt("Трасса 60")+"\n") //0
```

parseFloat

Преобразовать любое значение в число с плавающей точкой. Также действует с объектами (например **Math**, **Number**, **String**, **Boolean**, **File**), делая их значения числом с плавающей точкой.

Синтаксис:

parseFloat(Value)

Возврат:

(Ч)

Параметры:

Value - **(Л)**, для преобразования. **ОП**.

Пример:

```
Debug.trace(parseFloat("8.5 m")) //8.5
```

Операции

Приоритет операций.

Действия с операциями более высокого приоритета будут происходить раньше чем с остальными.
Операции [*] и [/] имеют одинаковый приоритет.
Операции [+] и [-] имеют одинаковый приоритет.
Операции [*] и [/] имеют больший приоритет, чем операции [+] и [-].
Операция [&&] имеет больший приоритет, чем [||].
Операции сравнения имеют больший приоритет, чем побитовые.
Выражения в круглых скобках () вычисляются раньше других операций.
Выражения в скобках можно заключать в другие скобки.
Все скобки в выражении должны быть закрыты, иначе появится сообщение об ошибке.

Арифметические операции.

[+] - сложение, например: **Debug.trace(3+2)**
[-] - вычитание, например: **Debug.trace(6-4)**
[*] - умножение, например: **Debug.trace(2*2)**
[/] - деление, например: **Debug.trace(9/3)**
[++] - инкремент (прибавление единицы), например: **x++** или **x=++x** (не **x=x++**) равносильны **x=x+1**
[--] - декремент (вычитание единицы), например: **x--** или **x=--x** (не **x=x--**) равносильны **x=x-1**
[%] - модуло (остаток от деления), например:

```
v=8%3;w=8%(-3);x=(-8)%3;y=(-8)%(-3);z=9%3
Debug.trace("v="+v+" w="+w+" x="+x+" y="+y+" z="+z) //результат: v=2 w=2 x=-2 y=-2 z=0
```

[^] не предназначен для возведения в степень, поэтому **x=y^2** неверно, а верно будет так: **x=Math.pow(y, 2)**

Операции присваивания.

Такие операции присваивают значение левой стороне от знака на основе значения правой стороны от знака. **ОЧЕНЬ ВАЖНО** запомнить разницу между операцией присваивания [=] и операцией сравнения [==], когда переменной надо присвоить значение ставится один знак равенства, а когда надо сравнить значение переменной с другим значением, то ставится двойной знак равенства.
[=] - присваивание, например: **x=10 //переменной x присвоено значение 10**
[+=] - добавление к, например: **x+=y** равносильно **x=x+y**
[-=] - вычитание из, например: **x-=y** равносильно **x=x-y**
[*=] - умножение на, например: **x*=y** равносильно **x=x*y**
[/=] - деление на, например: **x/=y** равносильно **x=x/y**
[%=] - присваивание с модуло, например: **x%=y** равносильно **x=x%y**
[&=] - присваивание с побитовым **И** (**AND**), например: **x&=y** равносильно **x=x&y**
[|=] - присваивание с побитовым **ИЛИ** (**OR**), например: **x|=y** равносильно **x=x|y**
[^=] - присваивание с побитовым **Исключающим ИЛИ** (**XOR**), например: **x^=y** равносильно **x=x^y**
[>>=] и [<<=] - присваивание с побитовым сдвигом, например: **x>>=y** равносильно **x=x>>y**

Операции сравнения.

Сравнивается значение с левой стороны от знака со значением с правой стороны от знака и возвращается булево значение **true** или **false**, основанное на верности сравнения.
[==] - равно, например: **Debug.trace((2==2)+" "+(2=="2")+" "+(3==2)) //true true false**
[===] - строгое равно, например: **Debug.trace((2===2)+" "+(2==="2")) //true false**
[!=] - не равно, например: **Debug.trace((3!=2)+" "+("A"!="B")+" "+("A"+"B"!="AB")) //true true false**
ВАЖНО: не равно вида [<>] работать не будет и вызовет ошибку.
[<] - меньше, например: **Debug.trace(2<3) //true**
[>] - больше, например: **Debug.trace(3>2) //true**
[<=] - меньше или равно, например: **Debug.trace((2<=2)+" "+(2<=3)) //true true**
[>=] - больше или равно, например: **Debug.trace((2>=2)+" "+(2>=3)) //true false**

Строковые операции.

[+] - объединение (сцепление) строк, например: **x="Arabian "+"Nights"** равносильно **x="Arabian Nights"**
[+=] - добавление к строке, например: **x="Arabian ";x+="Nights"** равносильно **x="Arabian Nights"**
НЕЛЬЗЯ разъединять строковые значения и отбавлять от них с помощью операций [-] и [-=].
Сравнение строковых значений с помощью операций [<] и [>] происходит посимвольно, с учетом алфавитного порядка, причем количество символов играет второстепенную роль, например: **"Z"** будет больше **"AB"**, **"ZA"** будет больше **"Z"**. Действует и с русскими буквами.

Логические операции.
Используются для проверки нескольких выражений, возвращают булево значение.
[**&&**] - логическое **И** (**AND**), возвращает **true** если каждое выражение истинно, иначе **false**, например:
x=0, y=0, z=1;Debug.trace(x==0 && y==0 && z==0) //false
[||] - логическое **ИЛИ** (**OR**), возвращает **true** если хотя бы одно выражение истинно, иначе **false**, например:
x=0, y=0, z=1;Debug.trace(x==0 || y==0 || z==0) //true
[^] - логическое **Исключающее ИЛИ** (**XOR**), возвращает **1** если только одно выражение истинно, иначе **0**, например:
x=0, y=0, z=1;Debug.trace(x==1 ^ y==1 ^ z==1) //1
[**!**] - логическое **НЕ** (**NOT**), возвращает противоположное булево значение, например:
Debug.trace(!false) //true
Побитовые операции.
Используются с числовыми значениями. Каждое из числовых значений переводится в двоичную систему счисления. Полученные двоичные числа сравниваются побитово и в результате возвращается числовое значение в десятичной системе счисления.
[**&**] - побитовое **И** (**AND**), например: **Debug.trace(11&13) //9 (1011 AND 1101 = 1001)**
[|] - побитовое **ИЛИ** (**OR**), например: **Debug.trace(11|13) //15 (1011 OR 1101 = 1111)**
[^] - побитовое **Исключающее ИЛИ** (**XOR**), например: **Debug.trace(11^13) //6 (1011 XOR 1101 = 0110)**
[~] - побитовое **НЕ** (**NOT**) - **не работает**.
При использовании операций побитовых сдвигов, числовое значение с левой стороны от знака переводится в двоичную систему счисления, а затем сдвигается на столько, сколько указано в целом числе с правой стороны от знака. Результат возвращается в десятичной системе счисления.
[<<] - побитовый левый сдвиг, например:
Debug.trace(11<<1) //сдвиг 11 (1011) влево, выводится 22 (10110)
[>>] - побитовый правый сдвиг, например:
Debug.trace(11>>1) //сдвиг 11 (1011) вправо, выводится 5 (101)

Первое значение	Второе значение	&	|	^
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Тернарная операция.
[**? :**] - присваивает одно из двух значений, в зависимости от истинности, проверяемого выражения.
Синтаксис:
varName=(expression)?valueIfTrue:valueIfFalse
Пояснение:
varName - переменная, которой присваивается значение.
expression - выражение для проверки на истинность.
valueIfTrue - значение, присваиваемое переменной, если проверяемое выражение истинно.
valueIfFalse - значение, присваиваемое переменной, если проверяемое выражение ложно.
Пример:

```
var a=1, b, c //объявлены три переменных, первой присвоено значение
b=(a>0)?"Правда":"Ложь" //переменной b присвоится значение "Правда"
c=(a<0)?"Правда":"Ложь" //переменной c присвоится значение "Ложь"
Debug.trace("b="+b+"\n"+"c="+c)
```

Операция разделения инструкций.
[**;**] - (точка с запятой) разделитель инструкций в коде, например: **a=0;b=1**
После каждой инструкции должна стоять точка с запятой для отделения одной инструкции от другой. Необязательно использовать разделитель если инструкции расположены на разных строчках. Если же писать инструкции в одну строчку то использование разделителя обязательно.
Операция последовательного вычисления.
[**,**] - (запятая) последовательное вычисление значений выражений в скобках, например:

```
a=5, b=4
c=(d=(a+b),a-b)
Debug.trace("c="+c+"\n"+"d="+d) //c=1, d=9
```

Переменные

Переменные являются контейнерами информации и должны быть объявлены (определены) перед их использованием.

Синтаксис:

var variableName=value

или:

variableName=value

Пояснение:

var - инструкция объявления переменной (инициализация может выполняться во время объявления). Значение объявленной, но не инициализированной переменной равно **null**.

1) Когда **var** используется **внутри функции**, переменная становится **локальной для этой функции** и не может использоваться вне функции.

2) Если переменная объявляется без инструкции **var**, то она становится свойством страницы и может использоваться вне функции или в другой функции.

3) При объявлении переменной без инструкции **var** необходимо провести инициализацию сразу.

4) Все переменные вне функций являются свойствами страницы.

variableName - название переменной, заданное пользователем.

1) Название переменной должно состоять только из латинских букв и цифр. Допустимыми специальными символами в названии являются: подчёркивание [_] и доллар [$]. Пробелов в названии быть не должно.

2) Название переменной должно начинаться только с буквы или допустимого специального символа. С цифры начинать нельзя.

3) Регистр буквенных символов влияет на название переменной, например **amount** и **Amount** будут двумя разными переменными (не рекомендуется этим пользоваться во избежание путаницы).

Примеры корректных названий для переменной: **Amount01**, **Amount_01**, **$Amount01**, **_01Amount**.

Примеры **НЕ**корректных названий для переменной: **01Amount**, **Amount 01**.

value - **(Л)**, присваиваемое значение переменной (может быть выражением).

Примеры:

Объявление переменной с инициализацией без инструкции var.

Variable1=12

Объявление нескольких переменных с инициализацией без инструкции var.

Variable1=12, Variable2="AB", Variable3=true

Объявление переменной.

var Variable1

Инициализация уже объявленнной переменной.

Variable1=12

Объявление нескольких переменных.

var Variable1, Variable2, Variable3

Инициализация нескольких уже объявленных переменных.

Variable1=12, Variable2="AB", Variable3=true

Объявление нескольких переменных вместе с инициализацией.

var Variable1=12, Variable2="AB", Variable3=true

Системные переменные

Значения этих переменных зависят от системы, на которой запущена публикация и содержат информацию о системе. Фактически системным переменным можно присвоить новые значения (не рекомендуется), но это не повлияет на то, что они обозначают (например системное время или разрешение экрана не поменяется). Если во время работы публикации системным переменным были присвоены новые значения, то нельзя будет сбросить новые значения на истинные до конца работы публикации.

Переменные системных путей:

SYSTEM_PUBLICATION_DIR - **(С)**, расположение текущей публикации (**null**, если публикация не сохранена).
SYSTEM_WIN_DIR - **(С)**, расположение директории **Windows**, установленной на компьютере.
SYSTEM_WINSYS_DIR - **(С)**, расположение системной директории **Windows**.
SYSTEM_PROGRAMS_DIR - **(С)**, расположение директории **Program Files**.
SYSTEM_PROGRAMDATA_DIR - **(С)**, расположение директории **Program Data** (в **Vista** и выше).
SYSTEM_DOCUMENTS_DIR - **(С)**, расположение директории **Мои документы** текущего пользователя.
SYSTEM_TEMP_DIR - **(С)**, расположение директории **Temp** текущего пользователя.
Все системные переменные путей уже содержат в конце обратный слэш, поэтому для продолжения указания конкретного пути, два обратных слэша в начале строкового значения ставить необязательно. Например, с помощью скрипта можно указать путь к нужному файлу публикации двумя способами:
SYSTEM_PUBLICATION_DIR+"\\DATA\\README.TXT"
SYSTEM_PUBLICATION_DIR+"DATA\\README.TXT"

Переменные информации о системе:

SYSTEM_CD_DRIVE - **(С)**, буквенное обозначение первого **CD/DVD** устройства.
SYSTEM_HAS_SOUND - **(Б)**, наличие звуковой карты (**true** - есть звуковая карта, иначе **false**).
SYSTEM_COLOUR_DEPTH - **(Ц)**, глубина цвета экрана.
SYSTEM_SCREEN_RES_X - **(Ц)**, ширина разрешения экрана.
SYSTEM_SCREEN_RES_Y - **(Ц)**, высота разрешения экрана.
SYSTEM_USERNAME - **(С)**, имя текущего пользователя.
SYSTEM_OPERATING_SYS - **(С)**, тип текущей операционной системы **Windows**, возможные значения: **"NT3"**, **"NT4"**, **"ME"**, **"95"**, **"98"**, **"2000"**, **"2003"**, **"XP"**, **"Vista"**, **"Windows 7"**, **"Windows 8"**.

Переменные системной даты и времени:

SYSTEM_DATE_FULL - **(С)**, полная текущая системная дата (день, месяц, год).
SYSTEM_TIME_YEAR - **(Ц)**, текущий год (четырехзначный) системной даты.
SYSTEM_TIME_MONTH - **(С)**, текущий месяц (трехбуквенный) системной даты.
SYSTEM_TIME_DATE - **(С)**, текущее число (день месяца) системной даты.
SYSTEM_TIME_DAY - **(С)**, текущий день недели (трехбуквенный) системной даты.
SYSTEM_TIME_HOUR - **(Ц)**, текущий час (от **0** до **23**) системного времени.
SYSTEM_TIME_12HOUR - **(Ц)**, текущий час (от **1** до **12**) системного времени.
SYSTEM_TIME_AMPM - **(С)**, до полудня (**"AM"**) или после полудня (**"PM"**) системного времени.
SYSTEM_TIME_MINUTE - **(Ц)**, текущие минуты системного времени.
SYSTEM_TIME_SECOND - **(Ц)**, текущие секунды системного времени.

Переменные сведений о публикации:

PUBLICATION_TITLE - **(С)**, название текущей публикации, заданное в органайзере (не название файла публикации).
PUBLICATION_PAGE_TITLE - **(С)**, название текущей страницы, заданное в органайзере.
PUBLICATION_TIME - **(Ц)**, количество секунд прошедших с момента запуска публикации.
COMMAND_PARAM_COUNT - **(Ц)**, количество параметров, введенных в командной строке (например: **start.exe /a /b**).
COMMAND_PARAM_# - **(С)**, значение параметра введённого в командной строке. Отдельно создается для каждого параметра (**#** - номер параметра, нумерация начинается с **1**).
PUBLICATION_EVALUATION - **(Ц)**, состояние пробного периода публикации. Возможные значения:
1 и более - количество дней оставшегося пробного периода.
0 - срок пробного периода истек.
-1 - публикация без пробного периода.
-2 - публикация была с пробным периодом, который отменён действием **Change Publication Evaluation**.
CHAPTER_PASSWORD - **(С)**, пароль ограничения доступа к текущей главе. Используется только в том случае, если установлены пароли для глав на вкладке **"Password"** диалога **"Publication Properties"**.

Объекты

Все компоненты, добавляемые на страницу в публикации (включая страницы, главы и саму публикацию) являются объектами. С помощью скрипта можно создавать объекты с различными данными.

1) **Нельзя** создавать объекты страницы, главы и публикации с помощью скрипта.

2) **Нельзя** создавать графические объекты с помощью скрипта, но есть возможность создавать клоны существующих графических объектов с помощью специальной функции.

3) На все объекты **кроме объектов страницы, главы и публикации** можно ссылаться в скрипте через их название, заданное в органайзере. На объекты страницы, главы и публикации можно ссылаться с помощью функций базовых объектов.

4) Абсолютно любому объекту можно создать новые свойства.

Синтаксис:

Создание объекта.

```
var varName=new objectType(parameters)
```

Применение функции к объекту.

```
objectName.function()
```

Создание свойства для объекта.

```
objectName.property=propertyValue
```

либо

```
objectName[propertyName]=propertyValue
```

Пояснение:

var - функция объявления новой переменной.

varName - название переменной (заданное пользователем) для хранения нового объекта, то же, что и название нового объекта.

new - инструкция, создающая новый экземпляр указанного типа объекта.

objectType - тип создаваемого объекта из возможных:

1) **Object** - объект хранения свойств;
2) **Array** - массив;
3) **Boolean** - логический объект;
4) **Number** - числовой объект;
5) **String** - строковый объект;
6) **Date** - объект даты и времени;
7) **Database** - объект базы данных.

parameters - необходимые параметры (если предусмотрены).

objectName - название объекта (заданное пользователем).

Не распознаются названия объектов, содержащие пробелы. Также нельзя использовать в названии объектов некоторые символы (например кавычки и символы, обозначающие операции). Но в органайзере любой объект можно переименовать как угодно, используя запрещенные символы, а также давая объектам названия зарезервированных ключевых слов и функций. Этого делать настоятельно не рекомендуется. По умолчанию в органайзере всем вновь созданным объектам даются названия, содержащие пробел, следующего вида **Object #** где **Object** - тип объекта, а # - номер объекта, например **Vector 1**. Чтобы в скрипте сослаться на такой объект, не переименовывая его в органайзере, следует заменить пробел в названии символом подчеркивания [_], например **Vector_1**, иначе объект не будет распознан скриптом. Лучше всего сразу переименовывать объекты в органайзере, убирая пробелы.

[**.**] - полная остановка, точка-разделитель между названием переменной и названием функции.

function() - название функции, применяемой к объекту. Может быть как одной из встроенных, так и созданной пользователем.

property - свойство объекта (свойства лучше называть строчными буквами, чтобы не путать с функциями).

propertyName - **(С)**, название свойства объекта.

propertyValue - **(Л)**, значение свойства объекта.

Пример:

```
var Jetpack=new Object() //создание нового объекта хранения свойств
Jetpack.level="Medium" //создание нового свойства для объекта первым способом
Jetpack["fuel"]=100 //создание нового свойства для объекта вторым способом
//вывод списка функций, возможных для применения к объекту и свойств объекта со значениями
for (Props in Jetpack) {Debug.trace(Props+": "+Jetpack[Props]+"\n")}
```

Логические (булевы) объекты

Логические объекты могут хранить только два значения: **true** (истина) или **false** (ложь).

ВАЖНО: Переменные ниже содержат логический объект с булевым значением, а не само булево значение. Для преобразования логического объекта внутри переменной в булево значение используется функция **valueOf**.

1) Любой из следующих способов можно использовать для создания нового логического объекта с начальным значением **false**:

```
var BoolF=new Boolean(false)
var BoolF=new Boolean(0)
var BoolF=new Boolean("")
var BoolF=new Boolean()
var BoolF=new Boolean(null)
var BoolF=new Boolean(undefined)
```

2) Любой из следующих способов можно использовать для создания нового логического объекта с начальным значением **true**:

```
var BoolT=new Boolean(true)
var BoolT=new Boolean(1)
var BoolT=new Boolean("любая непустая строка")
//Примеры непустых строк:
var BoolT=new Boolean(" ")
var BoolT=new Boolean("true")
var BoolT=new Boolean("false")
```

Массивы

Массив - ряд переменных, которые можно считывать по отдельности. Каждая переменная в ряду называется элементом массива. Получить доступ к любому элементу можно по его номеру в массиве. Для создания многомерных массивов (матриц) следует внутри основного массива в качестве элементов использовать другие массивы.

Синтаксис:

Создать (объявить) массив:

var arrayName=new Array(amount)

Присвоить значение элементу массива:

arrayName[elementNumber]=elementValue

Получить количество элементов в созданном массиве:

arrayName.length

Обратить порядок значений элементов в созданном массиве:

arrayName.reverse()

Сортировать значения элементов созданного массива в алфавитном порядке:

arrayName.sort()

Преобразовать значения всех элементов созданного массива в одно строковое значение:

arrayName.join(separator)

Разбить строковое значение на элементы массива (все элементы примут строковое значение):

stringValue.split(separator)

Пояснение:

var - функция, используемая для объявления и инициализации новой переменной.
arrayName - название массива, заданное пользователем.
new Array() - инструкция создания нового массива (**new** для создания нового объекта, **Array** для указания, что тип нового объекта - массив).
amount - **(Ц)**, количество элементов в массиве (если не указано, то неопределенное).
() - количество элементов массива должно быть заключено в круглые скобки.
elementNumber - **(Ц)**, номер элемента в массиве (нумерация начинается с **0**).
[ ] - номер элемента должен быть заключен в квадратные скобки.
elementValue - **(Л)**, значение элемента массива (может быть другим массивом).
length - свойство объекта массива, содержащее количество элементов.
reverse() - функция перестановки значений элементов массива в обратном порядке.
sort() - функция сортировки значений элементов массива в алфавитном порядке.
join() - функция преобразования элементов массива в строковое значение.
split() - функция разбиения строкового значения на элементы массива.
stringValue - **(С)**, для разбиения на элементы массива.
separator - **(С)**, обозначение разделительных символов для строки/массива. **НП**, по умолчанию:
для **join()** - запятая **","**;
для **split()** - без разделителя (строковое значение целиком будет единственным элементом массива).

Примеры:

Создание (объявление) массива.

```
var Colors=new Array(5) //создание масива, в котором будет пять элементов
или
var Colors=new Array() //создание пустого массива для заполнения по мере надобности
```

Заполнение элементов массива.

```
Colors[0]="Red"
Colors[1]="Green"
Colors[2]="Blue"
Colors[3]="Black"
Colors[4]="White"
```

Количество, обратный порядок и сортировка по алфавиту элементов массива.

```
//Подсчёт количества элементов в массиве
Debug.trace("Количество элементов в массиве: "+Colors.length)
//Присвоение значений элементам массива в обратном порядке от исходных
Colors.reverse();Debug.trace("\n"+Colors[4]) //последний элемент массива принял значение первого
//Присвоение значений элементам массива в алфавитном порядке
Colors.sort();Debug.trace("\n"+Colors[0]) //первый элемент массива принял значение "Black"
```

Создание массива из строкового значения и создание строкового значения из массива.

```
var String1="A-B-C-D-E-F"
var Array1=String1.split("-") //создание массива из строкового значения через переменную
var Array2="1,2,3,4,5,6,7,8,9".split(",") //создание массива из строкового значения напрямую
//Вывод в скриптовую консоль строкового значения "D6"
Debug.trace(Array1[3]+Array2[5])
//Вывод в скриптовую консоль строкового значения из объединенных элементов массива
Debug.trace("\n"+Array1.join("-&-"))
```

Массив как элемент другого массива.

```
var MainArray=new Array() //создание основного массива
var InnerArray=new Array() //создание массива для вложения в основной
MainArray[0]=InnerArray //первому элементу основного массива присваивается вложенный массив
InnerArray[0]="A" //первому элементу вложенного массива присваивается строковое значение
InnerArray[1]=8.25 //второму элементу вложенного массива присваивается числовое значение
InnerArray[2]=true //третьему элементу вложенного массива присваивается булево значение
Debug.trace(MainArray[0][1]) //вывод в скриптовую консоль второго элемента вложенного массива
```

Условия с помощью if и else (если истина, то ..., иначе ...)

Инструкция **if** выполняет набор инструкций, если проверяемое выражение (может состоять из нескольких выражений) истинно. Инструкция **else** может быть добавлена к инструкции **if**, и выполняет альтернативный набор инструкций, если проверяемое выражение ложно. Инструкция if может быть вложена в другую инструкцию **if** или **else**.

Синтаксис:

1) Обычное условие. Если выражение истинно, то выполняется набор инструкций:

if (expression) {list of statements for true}

2) Условие с альтернативой. Если выражение истинно, то выполняется набор инструкций, иначе выполняется альтернативный набор инструкций:

if (expression) {list of statements for true} else {list of statements for false}

Пояснение:

if - ключевое слово, используемое для выполнения фрагмента кода если выражение истинно.

else - ключевое слово, используемое для выполнения альтернативного фрагмента кода если выражение ложно.

expression - выражение, проверяемое на истинность.

list of statements for true - набор инструкций (фрагмент кода), выполняемый если выражение истинно.

list of statements for false - набор инструкций (фрагмент кода), выполняемый если выражение ложно.

() - проверяемое выражение должно быть заключено в круглые скобки.

{} - набор инструкций должен быть заключен в фигурные скобки.

Примеры:

Обычное условие (если истина, то...):

```
var CheckLetter="A"
if (CheckLetter=="A") {Debug.trace("Проверка успешна")}
```

Условие с альтернативой (если истина, то..., иначе...):

```
var CheckLetter="B"
if (CheckLetter=="A") {Debug.trace("Проверка успешна")} else {Debug.trace("Проверка провалена")}
```

Если проверяемое на истинность выражение возвращает булево значение, то необязательно использовать инструкцию сравнения в выражении, например:

```
//переменная B случайным образом будет иметь значение либо true, либо false
var A=new Boolean(Math.round(Math.random()));var B=A.valueOf();Debug.trace("B="+B+"\n")
//ниже представлены два аналогичных условия
if (B==true) {Debug.trace("Проверка успешна\n")} else {Debug.trace("Проверка провалена\n")}
if (B) {Debug.trace("Проверка успешна\n")} else {Debug.trace("Проверка провалена\n")}
```

Комплексное условие (отступы скобок используются для удобства чтения и необязательны):

```
var A=1, B=2, C=3 //правильные значения, с которыми можно поэкспериментировать
Debug.trace("Должно быть: A=1, B=2, C=3\n")
Debug.trace("Фактически: A="+A+", B="+B+", C="+C+"\n"+"Проверка:\n")
if (A==1) //начало комплексного условия
{ //открытие основного условия
Debug.trace("A-верно, ")
   if (B==2)
   { //открытие первого вложенного условия
  Debug.trace("B-верно, ")
       if (C==3)
       { //открытие второго вложенного условия
       Debug.trace("C-верно")
       } //закрытие второго вложенного условия
   } //закрытие первого вложенного условия
} //закрытие основного условия
else { //если A не равно 1 из основного условия, то выполняется код ниже, также с условиями
Debug.trace("A-неверно, ")
if ((B==2)&&(C==3)) {Debug.trace("B-верно, C-верно")}
if ((B!=2)&&(C!=3)) {Debug.trace("B-неверно, C-неверно")}
if (B==2&&C!=3) {Debug.trace("B-верно, C-неверно")}
if (B!=2&&C==3) {Debug.trace("B-неверно, C-верно")}
} //конец комплексного условия
```

Условия с помощью switch case (выбор варианта)

Функция **switch case** выполняет различные варианты наборов инструкций в зависимости от значения проверяемой переменной. Если ни один конкретный вариант (**case**) не соответствует значению проверяемой переменной, то выполняется набор инструкций особого варианта, называемого **default**. В отличие от **C++**, **case** может быть не только буквальным значением, но и выражением. Важно запомнить, что функция **"проваливается"**, т.е. после выполнения списка инструкций для конкретного варианта, функция перейдет к следующему варианту и выполнит также и его инструкции. Чтобы предотвратить **"провалы"** используется команда **break** после каждого варианта (ставить **break** после варианта **default** не обязательно, потому что он последний, но вариант не должен быть пустым). Если вариант не содержит инструкций, то команда **break** после него обязательна, иначе выведется сообщение об ошибке. Команда **break** завершает функцию **switch case** после выполнения инструкций варианта, соответствующего значению проверяемой переменной и не завершает функцию если вариант не соответствует значению проверяемой переменной, а происходит переход к следующему варианту. Вместо **break** в некоторых случаях можно использовать **return**, когда надо завершить не конкретно функцию **switch case**, а весь скрипт.

Синтаксис:

```
switch(variable) //проверка указанной переменной
{ //начало функции
case n: //вариант, выполняющийся если значение переменной равно n
list of statements //набор инструкций варианта n
break //переход к концу функции если значение переменной равно n
//другие варианты, если они есть, любое их количество
default: //вариант, выполняющийся при всех остальных значениях переменной
list of statements //набор инструкций варианта default
break
} //конец функции
```

Пример:

В данном примере демонстрируется использование выражений в вариантах и механизм действия **"провалов"**.

В начале необходимо присвоить переменной **x** значение от **1** до **10** (вместо **n**).

Если число четное, то в скриптовой консоли на выводе будет фрагмент или полный текст о волке.

Если число нечетное, то то в скриптовой консоли на выводе будет фрагмент или полный текст о зайце.

Чем меньше число, тем полнее будет текст и меньше заглушек в виде **"*"**.

Если значение **x** иное (например **0**, **-1**, **11**, **"лошадь"**), то на выводе будет слово **"КОНЕЦ"**.

```
var x=n //вместо n надо подставить число от 1 до 10, или что-то иное для варианта default
var y=12 //эта переменная введена для демонстрации использования выражений в вариантах
var a="*", b="*", c="*", d="*", e="*" //текстовые заглушки по умолчанию
switch (x) { //проверка переменной и начало функции
case 1: a="Волчок" //если сработает этот вариант, то на выводе будет полный текст о волке
case 3: b=" кусит" //если сработает этот вариант, то первое слово в тексте останется * и т.д.
case 5: c=" тебя"
case 7: d=" за"
case 9: e=" бочок"
break //переход к концу функции в случае успешного срабатывания одного из вариантов выше
case y-10: a="Заяц" //если сработает этот вариант, то на выводе будет полный текст о зайце
case y-8: b=" весело" //если сработает этот вариант, то первое слово в тексте останется * и т.д.
case y-6: c=" играет"
case y-4: d=" на"
case y-2: e=" барабанце"
break //переход к концу функции в случае успешного срабатывания одного из вариантов выше
default: a="К";b="О";c="Н";d="Е";e="Ц" //вариант выполнится при неправильном значении x
} //конец функции
Debug.trace (a+b+c+d+e) //вывод конечного текста в скриптовую консоль
```

Циклы for и while

Цикл повторно выполняет указанный фрагмент кода, пока проверяемое выражение цикла истинно, и прекращает выполнение данного фрагмента кода, если проверяемое выражение цикла становится ложным. Только после завершения работы цикла происходит переход к выполнению следующей строчки кода. Цикл **for** - со счётчиком, а цикл **while** - без счётчика. Для цикла **for** можно не использовать внутренний счётчик, а написать код собственного во фрагменте кода, как и для цикла **while**. С помощью специального цикла **for in** можно получить названия всех свойств объекта, и функций, возможных к применению на данный объект (число итераций у такого цикла будет равно сумме количества свойств объекта и количества функций для объекта).

Синтаксис:

```
for (initialise; expression; increment) {list of statements}
for (propVAR in Object) {list of statements}
while (expression) {list of statements}
```

Пояснение:

for и **while** - ключевые слова, обозначающие начало цикла.

initialise - выражение начального значения счётчика цикла **for**.

expression - проверяемое выражение цикла. Проверка данного выражения происходит в начале каждой итерации цикла. Если в результате проверки возвращается значение **true**, то выполняется заданный фрагмент кода и происходит переход к следующей итерации цикла, если **false**, то заданный фрагмент кода не выполняется, цикл завершается и просходит переход к следующей строчке кода.

increment - выражение изменения значения счётчика цикла for, выполняется в конце каждой итерации цикла.

list of statements - заданный фрагмент кода. Выполняется только если проверямое выражение цикла имеет значение **true**.

propVAR - переменная для хранения строковых значений названий свойств и функций объекта.

Object - объект, проверяемый на содерщащиеся в нём свойства, и функции, возможные к применению на него.

() - выражения цикла должны быть заключены в круглые скобки и разделены точкой с запятой.

{} - заданный фрагмент кода цикла должен быть заключен в фигурные скобки.

Любое из выражений (**initialise**, **expression**, **increment**) цикла **for** может не указываться, если стоят точки с запятой. Пропущенное выражение **expression** в цикле **for** считается **true**, поэтому цикл будет бесконечным, если не предусмотрен алгоритм выхода из цикла с помощью **break** или **return** во фрагменте кода цикла. В цикле **while** нельзя пропускать выражение **expression**, для создания бесконечного цикла следует указывать значение **true**. **ВАЖНО:** внутри бесконечного цикла всегда стоит делать небольшую паузу (хотя бы в тысячную долю секунды) с помощью функции **wait**, иначе программа может зависнуть.

Примеры:

Варианты применения разных циклов с одинаковым результатом выведения значений в консоль:

1) Обычное применение цикла for:

```
for (n=0; n!=11; n++) {Debug.trace(n+"\n")} //с прибавлением значения к счётчику
```

или

```
for (n=100; n!=(-10); n=n-10) {Debug.trace((10-n/10)+"\n")} //с отбавлением значения от счётчика
```

2) Бесконецный цикл for без внутреннего счётчика, а со счетчиком внутри фрагмента кода:

```
var n=0
for (;;)
{Debug.trace(n+"\n");n=n+1
wait(0.001) //"передышка" для процессора
if (n==1234) {break}}
```

3) Обычное применение цикла while:

```
var n=0; while (n!=11) {Debug.trace(n+"\n");n=n+1}
```

4) Бесконечный цикл while, завершающийся с помощью условия внутри фрагмента кода:

```
var n=0
while (true)
{Debug.trace(n+"\n");n=n+1
wait(0.001) //"передышка" для процессора
if (n==1234) {break}}
```

5) Цикл for in для перебора свойств объекта и возможных функций для него:

```
var Hero=new Object();Hero.chName="Conan";Hero.chClass="Barbarian";Hero.chLevel=35
for (Props in Hero) {Debug.trace(Props+": "+Hero[Props]+"\n")}
```

Остановка выполнения скрипта с помощью break, continue и return

break используется для выхода из циклов **for** и **while** или для завершения функции **switch case**. Строчки кода, идущие после цикла или функции **switch case** при использовании **break** будут выполнены.

continue используется в циклах **for** и **while** для перехода к следующей итерации, минуя выполнение кода внутри цикла, оставшегося после **continue**.

return используется для полного завершения скрипта, в котором вызывается, может применяться в условиях, циклах и пользовательских функциях. Строчки кода идущие после **return** не будут выполнены. Если из скрипта вызвана пользовательская функция, которая завершилась **return**, то выполнение этого скрипта продолжится после строчки вызова функции.

return может содержать выражение, значение которого нужно вернуть в основной скрипт. Данное выражение должно быть на одной строчке с **return**, а если выражение многострочное, то его следует поместить в круглые скобки, причем открывающая скобка должна быть на одной строчке с **return**.

Примеры:

Использование **break** продемонстрировано в примере к условию с помощью **switch case**.

***Использование* continue:**

```
var a=0
for (n=0; n<21; n++)
{if (a==1) {a=0; continue}
a=1
Debug.trace(n+"\n")} //вывод чисел не по порядку, а через одно
```

***Использование* return:**

В скриптовом объекте страницы **Script1** объявлена пользовательская функция **Tapirika**:

```
function Tapirika(ImgName) {ImgName.Move(100, 100, 3); return Name1="Чепрачный"}
```

Графический объект **Image1** при каком-то триггере выполняет скрипт:

```
var Name1="Малайский"
Tapirika(Image1)
Debug.trace("Тапир "+Name1+"\n")
```

1) Когда срабатывает триггер на объекте **Image1**, срабатывает скрипт прикрепленный к триггеру:
2) Объявляется переменная **Name1**, которой присваивается строковое значение **"Малайский"**.
3) На графический объект **Image1** действует пользовательская функция **Tapirika** из скрипта **Script1**.
4) Графический объект **Image1** перемещается на **100** пикселей вправо-вниз за **3** секунды.
5) Функция из скриптового объекта **Script1** останавливается с помощью **return**, причем при возвращении в скрипт триггера переменной **Name1** присваивается строковое значение **"Чепрачный"**.
6) В скриптовой консоли выводится **"Тапир Чепрачный"**.

В скриптовой консоли вывелось бы **"Тапир Малайский"** если бы функция **Tapirika** выглядела так:

```
function Tapirika(ImgName) {ImgName.Move(100, 100, 3); return Name1}
```

Функция wait

Функция **wait** используется для приостановки выполнения следующей строчки кода программы до тех пор, пока не истечет время (в секундах), заданное в функции. Эта функция полезна для упорядочивания нескольких событий, происходящих друг за другом. Время в функции можно установить на **0**, но это не бессмысленное применение функции, такой приём часто используется для обновления экрана.

Синтаксис:

```
wait(time)
```

Параметры:

time - **(Ч)**, время ожидания в секундах (как целое количество, так и доли). **ОП**.

Примеры:

Последовательность событий:

```
Image1.Show(); wait(1)
Image1.Hide(); wait(2)
Image2.Show(); wait(0.5)
Image2.Hide()
```

Обновление экрана:

В данном примере изображение очень быстро то появляется, то исчезает, бесконечное количество раз. Экран между появлением и исчезанием изображения не обновляется.

```
for(;;) {Image1.Show(); Image1.Hide()}
```

Для обновления экрана добавляется **wait(0)**:

```
for(;;) {Image1.Show(); wait(0); Image1.Hide(); wait(0)}
```

Математические функции

С помощью математических объектов **Math** и **Number** производятся математические вычисления. Все математические функции являются частью одного из математических объектов, одни из них - математические константы, другие - математические методы. Функции с математическими константами вводятся в верхнем регистре и не требуют ввода параметров, а функции с математическими методами вводятся в нижнем регистре и требуют ввода параметров.

Синтаксис:

var varName=mathObjName.functionName(Parameters)

Пояснение:

var - используется для создания переменной, в которой будет храниться математический объект.
varName - название переменной, заданное пользователем.
mathObjName - математический объект (либо **Math**, либо **Number**).
[**.**] - полная остановка, точка-разделитель между названием математического объекта и названием функции.
functionName() - название математической функции, которую требуется выполнить (у констант круглые скобки не ставятся).
Parameters - числовые параметры функции (функции с параметрами возвращают **NaN**, если в параметрах ничего не указано, либо указаны не числа).

Список функций:

Math.abs - получить модуль (абсолютное значение) числа.
Math.round - округлить число до ближайшего целого.
Math.ceil - округлить число в большую сторону.
Math.floor - округлить число в меньшую сторону.
Math.random - получить псевдослучайное число в промежутке от **0** до **1**.
Math.max - получить большее из двух чисел.
Math.min - получить меньшее из двух чисел.
Math.pow - возвести число в степень.
Math.exp - получить экспоненту (возвести число **E** в степень).
Math.atan2 - получить угол (в радианах) от **X**-оси к точке.
Math.sin - получить синус угла.
Math.asin - получить арксинус числа (в радианах).
Math.cos - получить косинус угла.
Math.acos - получить арккосинус числа (в радианах).
Math.tan - получить тангенс угла.
Math.atan - получить арктангенс числа (в радианах).
Math.log - получить натуральный логарифм числа.
Math.E - получить число **E** (приблизительно **2.718281828**).
Math.LN10 - получить натуральный логарифм **10** (приблизительно **2.302585093**).
Math.LOG2E - получить логарифм **E** по основанию **2** (приблизительно **1.442695041**).
Math.LOG10E - получить десятичный логарифм **E** (приблизительно **0.4342944819**).
Math.PI - получить число **Пи** (приблизительно **3.141592654**).
Math.sqrt - получить квадратный корень числа.
Math.SQRT1_2 - получить квадратный корень **0.5** (приблизительно **0.7071067812**).
Math.SQRT2 - получить квадратный корень **2** (приблизительно **1.414213562**).
Number.MAX_VALUE - получить наибольшее число (стремящееся к бесконечности).
Number.MIN_VALUE - получить наименьшее число (стремящееся к нулю).
Number.NEGATIVE_INFINITY - получить отрицательную бесконечность.
Number.POSITIVE_INFINITY - получить положительную бесконечность.
Number.NaN - получить не число **"Not-a-Number"**.

Math.abs

Получить модуль числа.

Синтаксис:

Math.abs(x)

Возврат:

(Ч), абсолютное (положительное) исходное.

Параметры:

x - **(Ч)**, исходное. **ОП**.

Пример:

```
Debug.trace("abs(-0.987)="+Math.abs(-0.987))
```

Math.round

Округлить число до ближайшего целого.

Синтаксис:

Math.round(x)

Возврат:

(Ц), округлённое в ближайшую сторону исходное.

Параметры:

x - **(Ч)**, исходное. **ОП**.

Пример:

```
Debug.trace("round(12.499)="+Math.round(12.499)+"\n") //округление до 12
Debug.trace("round(12.500)="+Math.round(12.500)) //округление до 13
```

Math.ceil

Округлить число до большего целого.

Синтаксис:

Math.ceil(x)

Возврат:

(Ц), округлённое в большую сторону исходное.

Параметры:

x - **(Ч)**, исходное. **ОП**.

Пример:

```
Debug.trace("ceil(12.111)="+Math.ceil(12.111)+"\n") //округление до 13
Debug.trace("ceil(-12.999)="+Math.ceil(-12.999)) //округление до -12
```

Math.floor

Округлить число до меньшего целого.

Синтаксис:

Math.floor(x)

Возврат:

(Ц), округлённое в меньшую сторону исходное.

Параметры:

x - **(Ч)**, исходное. **ОП**.

Примеры:

```
Debug.trace("floor(12.999)="+Math.floor(12.999)+"\n") //округление до 12
Debug.trace("floor(-12.111)="+Math.floor(-12.111)) //округление до -13
```

Math.random

Получить псевдослучайное число (сгенерированное компьютером по некоему алгоритму).

Синтаксис:

Math.random()

Возврат:

(Ч), псевдослучайное, в промежутке от **0** до **1**, с шестью знаками после запятой (например **0.358624**).

Пример:

```
Debug.trace("Случайное: "+Math.round(Math.random()*100)) //вывод случайного числа от 0 до 100
```

Math.max

Получить максимум.

Синтаксис:

Math.max(x, y)

Возврат: **(Ч)**, большее из двух исходных.

Параметры:

x - **(Ч)**, исходное. **ОП**.

y - **(Ч)**, исходное. **ОП**.

Пример:

```
Debug.trace("Максимум: "+Math.max(2, 8)) //нахождение максимума
```

Math.min

Получить минимум.

Синтаксис:

Math.min(x, y)

Возврат: **(Ч)**, меньшее из двух исходных.

Параметры:

x - **(Ч)**, исходное. **ОП**.

y - **(Ч)**, исходное. **ОП**.

Пример:

```
Debug.trace("Минимум: "+Math.min(2, 8)) //нахождение минимума
```

Math.pow

Возвести число в степень.

Синтаксис:

Math.pow(x, y)

Возврат:

(Ч), результат возведения в степень.

Параметры:

x - **(Ч)**, возводимое в степень. **ОП**.

y - **(Ч)**, показатель степени. **ОП**.

Пример:

```
Debug.trace("2^8="+Math.pow(2, 8)) //2 в степени 8
```

Math.exp

Получить экспоненту числа.

Синтаксис:

Math.exp(x)

Возврат:

(Ч), **E** (приблизительно **2.718281828**), возведенное в степень исходного.

Параметры:

x - **(Ч)**, исходное. **ОП**.

Пример:

```
Debug.trace("exp(8)="+Math.exp(8)) //возведение числа E в степень 8
```

Math.atan2

Получить угол в радианах между **X**-осью и вектором.

Синтаксис:

Math.atan2(y, x)

Возврат:

(Ч), угол в радианах между **X**-осью и вектором из точки **(0, 0)** к точке **(x, y)**.

Параметры:

y - **(Ч)**, **Y**-координата второй точки вектора. **ОП**.

x - **(Ч)**, **X**-координата второй точки вектора. **ОП**.

Пример:

```
Debug.trace("Угол между X-осью и вектором (2,8): "+Math.atan2(8,2)*180/Math.PI) //в градусах
```

Math.sin

Получить синус угла.

Синтаксис:

Math.sin(x)

Возврат:

(Ч), синус угла.

Параметры:

x - **(Ч)**, угол в радианах. **ОП**.

Пример:

```
Debug.trace("SIN 45 градусов: "+Math.sin(0.785398)) //0.785398 радиан = 45 градусов
```

Math.asin

Получить арксинус числа.

Синтаксис:

Math.asin(x)

Возврат:

(Ч), арксинус исходного (угол в радианах).

Параметры:

x - **(Ч)**, исходное. **ОП**.

Пример:

```
Debug.trace("ARCSIN 0.71: "+Math.asin(0.707107)*180/Math.PI) //в градусах
```

Math.cos

Получить косинус угла.

Синтаксис:

Math.cos(x)

Возврат:

(Ч), косинус угла.

Параметры:

x - **(Ч)**, угол в радианах. **ОП**.

Пример:

```
Debug.trace("COS 45 градусов: "+Math.cos(0.785398)) //0.785398 радиан = 45 градусов
```

Math.acos

Получить арккосинус числа.

Синтаксис:

Math.acos(x)

Возврат:

(Ч), арккосинус исходного (угол в радианах).

Параметры:

x - **(Ч)**, исходное. **ОП**.

Пример:

```
Debug.trace("ARCCOS 0.71: "+Math.acos(0.707107)*180/Math.PI) //в градусах
```

Math.tan

Получить тангенс угла.

Синтаксис:

Math.tan(x)

Возврат:

(Ч), тангенс угла.

Параметры:

x - **(Ч)**, угол в радианах. **ОП**.

Пример:

```
Debug.trace("TG 45 градусов: "+Math.tan(0.785398)) //0.785398 радиан = 45 градусов
Debug.trace("\n"+"CTG 45 градусов: "+Math.cos(0.785398)/Math.sin(0.785398))
```

Math.atan

Получить арктангенс числа.

Синтаксис:

Math.atan(x)

Возврат:

(Ч), арктангенс исходного (угол в радианах).

Параметры:

x - **(Ч)**, исходное. **ОП**.

Пример:

```
Debug.trace("ARCTG 1: "+Math.atan(1)*180/Math.PI) //в градусах
```

Math.log

Получить натуральный логарифм числа.

Синтаксис:

Math.log(x)

Возврат:

(Ч), натуральный логарифм исходного (по основанию **E**).

Параметры:

x - **(Ч)**, исходное. **ОП**.

Пример:

```
Debug.trace("LOG2 8="+Math.log(8)/Math.log(2)) //логарифм 8 по основанию 2
```

Math.E

Константа. Число **E**, основание натурального логарифма.

Синтаксис:

Math.E

Возврат: **(Ч)**, приблизительно **2.718281828**.

Math.LN10

Константа. Натуральный логарифм числа **10**.

Синтаксис:

Math.LN10

Возврат: **(Ч)**, приблизительно **2.302585093**.

Math.LOG2E

Константа. Логарифм **E** по основанию **2**.

Синтаксис:

Math.LOG2E

Возврат: **(Ч)**, приблизительно **1.442695041**.

Math.LOG10E

Константа. Логарифм **E** по основанию **10**.

Синтаксис:

Math.LOG10E

Возврат: **(Ч)**, приблизительно **0.4342944819**.

Math.PI

Константа. Число **Пи** (**π**), отношение длины окружности к диаметру.

Синтаксис:

Math.PI

Возврат: **(Ч)**, приблизительно **3.141592654**.

Пример:

```
var r=8 //радиус круга
Debug.trace("Площадь круга с радиусом "+r+": "+Math.round(Math.PI*Math.pow(r, 2)))
```

Math.sqrt

Получить квадратный корень числа.
Синтаксис:
Math.sqrt(x)
Возврат:
(Ч), квадратный корень исходного.
Параметры:
x - **(Ч)**, исходное. **ОП**.
Пример:

```
Debug.trace("sqrt(16)="+Math.sqrt(16))
```

Math.SQRT1_2

Константа. Квадратный корень числа **0.5**.
Синтаксис: **Math.SQRT1_2**
Возврат: **(Ч)**, приблизительно **0.7071067812**.

Math.SQRT2

Константа. Квадратный корень числа **2**.
Синтаксис: **Math.SQRT2**
Возврат: **(Ч)**, приблизительно **1.414213562**.

Number.MAX_VALUE

Константа. Наибольшее (стремящееся к бесконечности) возможное значение числа. Числа выше этого значения считаются бесконечностью.
Синтаксис: **Number.MAX_VALUE**
Возврат: **(Ч)**, **1.797693135e+308**.
Пример:

```
Debug.trace(Number.MAX_VALUE/Math.pow(10, 308))
```

Number.MIN_VALUE

Константа. Наименьшее (стремящееся к нулю) возможное значение числа. Числа ниже этого значения считаются **0**.
Синтаксис: **Number.MIN_VALUE**
Возврат: **(Ч)**, **2.225073859e-308**.
Пример:

```
Debug.trace(Number.MIN_VALUE/Math.pow(10, -308))
```

Number.NEGATIVE_INFINITY

Константа. Отрицательная бесконечность.
Синтаксис: **Number.NEGATIVE_INFINITY**
Возврат: **-Infinity**.
Пример:

```
Debug.trace("Минус бесконечность: "+Number.NEGATIVE_INFINITY)
```

Number.POSITIVE_INFINITY

Константа. Положительная бесконечность.
Синтаксис: **Number.POSITIVE_INFINITY**
Возврат: **Infinity**.
Пример:

```
Debug.trace("Бесконечность: "+Number.POSITIVE_INFINITY)
```

Number.NaN

Константа. Значение, которое не является числом (**Not-a-Number**).
Синтаксис: **Number.NaN**
Возврат: **NaN**.
Пример:

```
Debug.trace("Не число: "+Number.NaN)
```

Дата и время

Объект **Date** и связанные с ним функции предназначены для различных операций с датой и временем, и взаимодействия отдельно с такими свойствами, как: день, месяц, год, часы, минуты, секунды, миллисекунды. Функции с **UTC** в названии настраивают на всемирное координированное время (универсальное время).

Объект **Clock** и связанные с ним функции предназначены для создания таймеров в публикации. Таймер измеряет часы, минуты и секунды.

Синтаксис:

Создание нового объекта даты и времени:

var varName=new Date(Year, Month, Day, Hour, Minute)

или

var varName=new Date(FullDate)

Использование функций даты и времени:

varName.functionName

Пояснение:

var - функция объявления новой переменной.

varName - название переменной, заданное пользователем. В ней будут храниться дата и время.

new - инструкция, создающая новый экземпляр указанного объекта.

Date - объект даты и времени.

[**.**] - полная остановка, точка-разделитель между названием переменной и названием функции.

functionName - название функции даты и времени, которую требуется выполнить.

Параметры:

Year - **(Ц)**, год.

Month - **(Ц)**, номер месяца. Нумерация месяцев начинается с **0** (**0** - Январь, **11** - Декабрь).

Day - **(Ц)**, день.

Hour - **(Ц)**, час.

Minute - **(Ц)**, минуты.

FullDate - значение даты и времени. **НП**, по умолчанию текущие системная дата и время компьютера.

Может быть введено как:

(Ц), количество миллисекунд, прошедших с полуночи **01.01.1970**.

(С), по образцу **"13 Month 1990, 00:00:00"**.

Для ввода даты строкой разрешено использование как сокращенных английских названий месяцев, так и полных.

Сокращенные названия месяцев:

"Jan" - Январь.

"Feb" - Февраль.

"Mar" - Март.

"Apr" - Апрель.

"May" - Май.

"Jun" - Июнь.

"Jul" - Июль.

"Aug" - Август.

"Sep" - Сентябрь.

"Oct" - Октябрь.

"Nov" - Ноябрь.

"Dec" - Декабрь.

Список функций:

getFullYear - получить год.
setFullYear - установить год.
getMonth - получить месяц.
setMonth - установить месяц.
getDay - получить день недели.
getDate - получить число (день месяца).
setDate - установить число (день месяца).
getHours - получить час.
setHours - установить час.
getMinutes - получить минуты.
setMinutes - установить минуты.
getSeconds - получить секунды.
setSeconds - установить секунды.
getMilliseconds - получить миллисекунды.
setMilliseconds - установить миллисекунды.
getUTCFullYear - получить год по универсальному времени.
setUTCFullYear - установить год по универсальному времени.
getUTCMonth - получить месяц по универсальному времени.
setUTCMonth - установить месяц по универсальному времени.
getUTCDay - получить день недели по универсальному времени.
getUTCDate - получить число (день месяца) по универсальному времени.
setUTCDate - установить число (день месяца) по универсальному времени.
getUTCHours - получить час по универсальному времени.
setUTCHours - установить час по универсальному времени.
getUTCMinutes - получить минуты по универсальному времени.
setUTCMinutes - установить минуты по универсальному времени.
getUTCSeconds - получить секунды по универсальному времени.
setUTCSeconds - установить секунды по универсальному времени.
getUTCMilliseconds - получить миллисекунды по универсальному времени.
setUTCMilliseconds - установить миллисекунды по универсальному времени.
getTime - получить количество миллисекунд с **01.01.1970**.
setTime - установить дату и время с **01.01.1970** в миллисекундах.
Date.parse - получить из строкового значения количество миллисекунд с **01.01.1970**.
getTimezoneOffset - получить разницу между временем местного часового пояса и универсальным.
toLocaleString - преобразовать объект даты и времени в строку, используя время местного часового пояса.
toUTCString - преобразовать объект даты и времени в строку, используя универсальное время.
toGMTString - преобразовать объект даты и времени в строку, используя среднее время по Гринвичу.
CreateClock - создать новый объект таймера.
Start - запустить объект таймера.
Stop - остановить объект таймера.
Pause - поставить на паузу объект таймера.
Continue - продолжить, поставленный на паузу объект таймера.
GetClock - получить указанный объект таймера.
GetHours - получить час от объекта таймера.
GetMinutes - получить минуты от объекта таймера.
GetSeconds - получить секунды от объекта таймера.

getFullYear и getUTCFullYear

Получить от указанного объекта даты и времени текущий год.

Синтаксис:

```
getFullYear()
getUTCFullYear()
```

Возврат:

(Ц), текущий год (четырехзначный).

Пример:

```
var Date1=new Date()
Debug.trace("Текущий год: "+Date1.getFullYear())
```

setFullYear и setUTCFullYear

Установить для указанного объекта даты и времени нужный год.

Синтаксис:

```
setFullYear(Year)
setUTCFullYear(Year)
```

Параметры:

Year - **(Ц)**, четырехзначный год, больший чем **1970**. **ОП**.

Пример:

```
var Date1=new Date();Date1.setFullYear(1997)
Debug.trace("Год изменен: "+Date1.getFullYear())
```

getMonth и getUTCMonth

Получить от указанного объекта даты и времени текущий месяц.

Синтаксис:

```
getMonth()
getUTCMonth()
```

Возврат:

(Ц), номер месяца. Нумерация начинается с **0**.

Пример:

```
var Date1=new Date();var Month1=Date1.getMonth()
Debug.trace("Текущий месяц: "+Month1+1)
```

setMonth и setUTCMonth

Установить для указанного объекта даты и времени нужный месяц.

Синтаксис:

```
setMonth(Month)
setUTCMonth(Month)
```

Параметры:

Month - **(Ц)**, номер месяца. Нумерация начинается с **0**. Также для изменения не только месяца, но и года, используются числа больше **11** и отрицательные числа (например: **12** изменит дату на январь следующего года, а **-2** на декабрь предыдущего). **ОП**.

Пример:

```
var Date1=new Date()
Date1.setMonth(5) //изменить месяц на июнь
Debug.trace("Месяц изменен: "+Date1)
```

getDay и getUTCDay

Получить от указанного объекта даты и времени текущий день недели.

Синтаксис:

```
getDay()
getUTCDay()
```

Возврат:

(Ц), номер дня недели. Нумерация начинается с **0** (**0** - Воскресенье, **1** - Понедельник).

Пример:

```
var Date1=new Date(); Debug.trace("Текущий день недели: "+Date1.getDay())
```

getDate и getUTCDate

Получить от указанного объекта даты и времени текущее число (день месяца).

Синтаксис:

getDate()
getUTCDate()

Возврат:

(Ц), текущий день месяца (от **1** до **31**).

Пример:

```
var Date1=new Date(); var MDay1=Date1.getDate()
Debug.trace("Текущее число: "+MDay1)
```

setDate и setUTCDate

Установить для указанного объекта даты и времени нужное число (день месяца).

Синтаксис:

setDate(Date)
setUTCDate(Date)

Параметры:

Date - **(Ц)**, номер дня месяца. Нумерация дней начинается с **1**. Также для изменения не только числа, но и месяца используются числовые значения большие, чем количество дней в месяце, отрицательные числа и **0**. **0** и отрицательные числа переводят дату назад на количество указанных дней (например: **0** переведет на последнее число предыдущего месяца, а **-9** на десять дней назад). Значения числа большие чем положено в данном месяце, переводят дату вперед на количество дней указанных сверх положенного количества дней этого месяца). **ОП**.

Пример:

```
var Date1=new Date(); Date1.setDate(13)
Debug.trace("Число изменено: "+Date1)
```

getHours и getUTCHours

Получить от указанного объекта даты и времени текущий час.

Синтаксис:

getHours()
getUTCHours()

Возврат:

(Ц), текущий час (от **0** до **23**).

Пример:

```
var Date1=new Date(); var Hour1=Date1.getHours()
Debug.trace("Который час? "+Hour1)
```

setHours и setUTCHours

Установить для указанного объекта даты и времени нужный час.

Синтаксис:

setHours(Hours)
setUTCHours(Hours)

Параметры:

Hours - **(Ц)**, час. Нумерация часов начинается с **0**. Также для изменения не только часа, но и дня используются числовые значения большие, чем количество часов в дне и отрицательные числа. Отрицательные числа переводят дату назад на количество указанных часов (например: **-2** это **22** предыдущего дня). Значения числа большие чем **23**, переводят дату вперед на количество часов указанных сверх **23** (например: **24** это **0** следующего дня). **ОП**.

Пример:

```
var Date1=new Date(); Date1.setHours(14)
Debug.trace("Час изменен: "+Date1)
```

getMinutes и getUTCMinutes

Получить от указанного объекта даты и времени текущее количество минут.

Синтаксис:

getMinutes()
getUTCMinutes()

Возврат:

(Ц), текущее количество минут (от **0** до **59**).

Пример:

```
var Date1=new Date(); var Min1=Date1.getMinutes()
Debug.trace("Сколько минут? "+Min1)
```

setMinutes и setUTCMinutes

Установить для указанного объекта даты и времени нужное количество минут.

Синтаксис:

setMinutes(Minutes)
setUTCMinutes(Minutes)

Параметры:

Minutes - **(Ц)**, минуты. Нумерация минут начинается с **0**. Также для изменения не только минут, но и часа используются числовые значения большие, чем количество минут в часе и отрицательные числа. Отрицательные числа переводят дату назад на количество указанных минут (например: **-2** это **58** минут предыдущего часа). Значения числа большие чем **59**, переводят дату вперед на количество минут указанных сверх (например: **60** это **0** минут следующего часа). **ОП**.

Пример:

```
var Date1=new Date(); Date1.setMinutes(39)
Debug.trace("Минуты изменены: "+Date1)
```

getSeconds и getUTCSeconds

Получить от указанного объекта даты и времени текущее количество секунд.

Синтаксис:

getSeconds()
getUTCSeconds()

Возврат:

(Ц), текущее количество секунд (от **0** до **59**).

Пример:

```
var Date1=new Date(); var Sec1=Date1.getSeconds()
Debug.trace("Сколько секунд? "+Sec1)
```

setSeconds и setUTCSeconds

Установить для указанного объекта даты и времени нужное количество секунд.

Синтаксис:

setSeconds(Seconds)
setUTCSeconds(Seconds)

Параметры:

Seconds - **(Ц)**, секунды. Нумерация секунд начинается с **0**. Также для изменения не только секунд, но и минут используются числовые значения большие, чем количество секунд в минуте и отрицательные числа. Отрицательные числа переводят дату назад на количество указанных секунд (например: **-2** это **58** секунд предыдущей минуты). Значения числа большие чем **59**, переводят дату вперед на количество секунд указанных сверх (например: **60** это **0** секунд следующей минуты). **ОП**.

Пример:

```
var Date1=new Date(); Date1.setSeconds(48)
Debug.trace("Секунды изменены: "+Date1)
```

getMilliseconds и getUTCMilliseconds

Получить от указанного объекта даты и времени текущее количество миллисекунд.

Синтаксис:

getMilliseconds()
getUTCMilliseconds()

Возврат:

(Ц), текущее количество миллисекунд (от **0** до **999**).

Пример:

```
var Date1=new Date(); var MSec1=Date1.getMilliseconds()
Debug.trace("Сколько миллисекунд? "+MSec1)
```

setMilliseconds и setUTCMilliseconds

Установить для указанного объекта даты и времени нужное количество миллисекунд.

Синтаксис:

setMilliseconds(Milliseconds)
setUTCMilliseconds(Milliseconds)

Параметры:

Milliseconds - **(Ц)**, миллисекунды. Нумерация миллисекунд начинается с **0**. Также для изменения не только миллисекунд, но и секунд используются числовые значения большие, чем количество миллисекунд в секунде и отрицательные числа. Отрицательные числа переводят дату назад на количество указанных миллисекунд (например: **-2** это **998** миллисекунд предыдущей секунды). Значения числа большие чем **999**, переводят дату вперед на количество миллисекунд указанных сверх (например: **1000** это **0** миллисекунд следующей секунды). **ОП**.

Пример:

```
var Date1=new Date(); Date1.setMilliseconds(256)
Debug.trace("Миллисекунды изменены: "+Date1.getMilliseconds())
```

getTime

Получить дату и время в миллисекундах.

Синтаксис:

getTime()

Возврат:

(Ц), количество миллисекунд, прошедших с полуночи **01.01.1970** до даты и времени указанного объекта.

Пример:

```
var Date1=new Date(); Date1.setMilliseconds(256)
var Date2=new Date() //создание второго объекта даты со значением текущего времени
Debug.trace("Разница: "+(Date2.getTime()-Date1.getTime()))
```

setTime

Установить дату и время в миллисекундах.

Синтаксис:

setTime(Milliseconds)

Параметры:

Milliseconds - **(Ц)**, миллисекунды, прошедшие с полуночи **01.01.1970** до нужной даты и времени. **ОП**.

Пример:

```
var Date1=new Date(); Date1.setTime(Date.parse("3 Sep 2006, 19:08:43"))
Debug.trace("Новые дата и время: "+Date1)
```

Date.parse

Перевести текст строкового значения в значение даты и времени в миллисекундах, прошедших с полуночи **01.01.1970** до даты и времени, указанных в переводимом строковом значении.

Синтаксис:

Date.parse(Date)

Возврат:

(Ц), количество миллисекунд.

Параметры:

Date - **(С)**, которое будет переводиться в миллисекунды. **ОП**.

Пример:

```
var Date1=new Date()
var FromString=Date.parse("3 Sep 2006, 19:08:43") //перевести текст в миллисекунды
Date1.setTime(FromString) //установить переведенные миллисекунды в дату и время
Debug.trace("Время, конвертированное из текста: "+Date1)
```

getTimezoneOffset

Получить разницу в минутах между временем местного часового пояса и универсальным временем.

Синтаксис:

getTimezoneOffset()

Возврат:

(Ц), разница в минутах между временем местного часового пояса и универсальным временем.

Пример:

```
var Date1=new Date()
Debug.trace("Разница в минутах: "+Date1.getTimezoneOffset())
```

toLocaleString

Преобразовать в строку, переведенный во время местного часового пояса, объект даты и времени.

Синтаксис:

toLocaleString()

Возврат:

(С), среднее время местного часового пояса.

Пример:

```
var Date1=new Date()
Debug.trace("Время местного часового пояса: "+Date1.toLocaleString())
```

toUTCString

Преобразовать в строку, переведенный в универсальное время, объект даты и времени.

Синтаксис:

toUTCString()

Возврат:

(С), универсальное время.

Пример:

```
var Date1=new Date()
Debug.trace("Универсальное время: "+Date1.toUTCString())
```

toGMTString

Преобразовать в строку, переведенный в среднее время по Гринвичу, объект даты и времени.

Синтаксис:

toGMTString()

Возврат:

(С), среднее время по Гринвичу.

Пример:

```
var Date1=new Date()
Debug.trace("Среднее время по Гринвичу: "+Date1.toGMTString())
```

CreateClock и другие функции таймера

Создать таймер и измерять часы, минуты и секунды. Таймер может отображаться на странице внутри названной переменной.
Созданный таймер никогда не запустится без функции **Start**.

Синтаксис:

Создать объект таймера:
var ClockObj=CreateClock(ClockName, ClockVar, Format)
Запустить созданный объект таймера с нуля (даже после остановки или паузы):
ClockObj.Start()
Остановить объект таймера в его текущем времени без возможности продолжения:
ClockObj.Stop()
Остановить объект таймера в его текущем времени с возможностью продолжения:
ClockObj.Pause()
Продолжить работу таймера с момента остановки на паузу:
ClockObj.Continue()
Получить объект созданного таймера по его названию.
GetClock(ClockName)
(**Возврат:** **(О)**, таймер).
Получить количество часов, в течение которых воспроизводился объект таймера:
ClockObj.GetHours()
(**Возврат:** **(Ц)**, количество часов, прошедших со старта таймера).
Получить количество минут, в течение которых воспроизводился объект таймера:
ClockObj.GetMinutes()
(**Возврат:** **(Ц)**, количество минут, прошедших со старта таймера).
Получить количество секунд, в течение которых воспроизводился объект таймера:
ClockObj.GetSeconds()
(**Возврат:** **(Ц)**, количество секунд, прошедших со старта таймера).

Параметры:

ClockObj - **(О)**, таймер. **ОП**.
ClockName - **(С)**, название таймера. **ОП**.
ClockVar - **(С)**, название переменной отображения таймера на странице публикации. **ОП**.
Format - **(С)**, формат отображения таймера на странице. Формат часов: **[h]** или **[hh]**. Формат минут: **[m]** или **[mm]**. Формат секунд: **[s]** или **[ss]**. Также к формату могут быть добавлены другие символы. **НП**, по умолчанию: **"[hh]:[mm]:[ss]"**.

Пример:

```
var Timer1VAR
var Timer1Format="[hh] ч [mm] мин [ss] сек"
var Clock1=CreateClock("Timer1", "Timer1VAR", Timer1Format)
Clock1.Start()
Clock1.Stop()
Clock1.Start();wait(3)
Clock1.Pause()
Clock1.Continue();wait(3)
Debug.trace(Clock1.GetHours()+"ч "+Clock1.GetMinutes()+"м "+Clock1.GetSeconds()+"с")
//текстовый объект Text1 должен быть создан на странице
Text1.SetSelection(0, -1);Text1.ReplaceSelection(Timer1VAR)
var Clock2=GetClock("Timer1")
Debug.trace("\nТолько секунды: "+Clock2.GetSeconds()+"с")
```

Строковые объекты

1) Глобальный строковый объект **String** предназначен для операций с различными типами данных и работает с ними как со строковыми.
Возвращает строковое значение (кроме функций преобразования данных).
2) Строковый объект **String** преобразует любой тип данных в строковое значение и хранит его в себе.
Возвращает объект со строковым значением, а не просто строковое значение.
ВАЖНО: Переменная, которой присвоено строковое значение - это **НЕ** то же самое, что переменная которой присвоено значение строкового объекта со строковым значением.
1) Создание переменной со значением, преобразованным с помощью использования функции глобального строкового объекта:
var varName=String.GlobalStringFunctionName(Value, parameters)
2) Создание переменной, содержащей строковый объект:
var varName=new String(Value)
3) Создание переменной и присвоение ей значения (например строкового):
var varName=Value
4) Использование функции строкового объекта (или значения):
varName.StringFunctionName(parameters)
5) Использование глобальных функций для строковых объектов и значений:
GlobalFunctionForString(Value, parameters)
Пояснение:
var - инструкция объявления новой переменной.
new - инструкция создания нового экземпляра указанного типа объекта.
varName - название переменной, заданное пользователем.
String - обозначение глобального строкового объекта или обычного строкового объекта.
[**.**] - полная остановка, точка-разделитель между объектом (переменной) и названием применяемой функции.
GlobalStringFunctionName - название выполняемой функции глобального строкового объекта.
StringFunctionName - название выполняемой функции строкового объекта.
GlobalFunctionForString - название глобальной функции для строковых объектов и значений.
Value - **(Л)** для функций глобального строкового объекта и глобальных строковых функций, **(С)** для функций строковых объектов и значений.
parameters - параметры функции (если есть).
Список функций глобального строкового объекта:
String.length - получить длину строкового значения.
String.mid - получить выбранную часть строкового значения.
String.left - получить левую часть строкового значения.
String.right - получить правую часть строкового значения.
String.toupper - преобразовать все символы строкового значения в символы верхнего регистра.
String.tolower - преобразовать все символы строкового значения в символы нижнего регистра.
String.contains - проверить содержание одного строкового значения внутри текста другого.
String.word - получить слово из строкового значения.
String.fromCharCode - пребразовать **ASCII**-код в символ.
String.random - получить случайное число как строковое значение.
String.format - назначить формат отображения числа как строкового значения.
String.bool - преобразовать любое значение в булево.
String.integer - преобразовать любое значение в целое число.
String.number - преобразовать любое значение в числовое.
String.string - преобразовать любое значение в строковое.
Список глобальных строковых функций:
escape - закодировать специальные символы в строковом значении.
unescape - вернуть в исходное состояние строковое значение с закодированными специальными символами.
CountLines - получить количество строчек в строковом значении.
CountWords - получить количество слов в строковом значении.
StringReplace - получить строковое значение из любого с заменой одинаковых фрагментов текста на другой.
PrintString - распечатать строковое значение.
ToClipboard - скопировать любые данные в буфер обмена.
GenerateGUID - сгенерировать строковое значение с глобально-уникальным идентификатором.

Список функций строковых объектов и значений:
length - получить длину строкового значения.
charAt - получить символ из строкового значения.
charCodeAt - пребразовать символ в **ASCII**-код.
toUpperCase - преобразовать все символы строкового значения в символы верхнего регистра.
toLowerCase - преобразовать все символы строкового значения в символы нижнего регистра.
substring - получить указанную часть строкового значения.
split - разбить строковое значение на элементы массива.
indexOf - найти с левого конца номер позиции первого символа строкового значения внутри другого.
lastIndexOf - найти с правого конца номер позиции первого символа строкового значения внутри другого.

String.length

Определить количество символов в строковом значении.
Синтаксис:
String.length(String)
Возврат:
(Ц), количество символов (включая пробелы) в строковом значении.
Параметры:
String - **(Л)**, исходное. **ОП**.
Пример:
```
Debug.trace(String.length("Лабиринт минотавра")+"\n") //результат: 18
Debug.trace(String.length(9999)+"\n") //результат: 4
Debug.trace(String.length(Vector1)) //результат: 19 (длина названия объекта "[ILMObject Vector1]")
```

String.mid

Получить одно строковое значение из указанной части другого.
Синтаксис:
String.mid(String, Index, Chars)
Возврат:
(С), состоящее из заданного количества символов, начинающееся с указанного символа исходного строкового значения.
Параметры:
String - **(Л)**, исходное. **ОП**.
Index - **(Ц)**, порядковый номер (нумерация начинается с **1**) символа в исходном строковом значении, с которого будет начинаться новое строковое значение. **ОП**.
Chars - **(Ц)**, количество, взятых из исходного строкового значения, символов, начиная с символа по указанным номером. **НП**, по умолчанию остаток исходного строкового значения, начинающийся с символа под указанным номером (такой же результат, если берётся количество символов, превышающее возможное для взятия).
Пример:
```
Debug.trace(String.mid("Старый Новый год", 8, 5)) //результат: "Новый"
```

String.left

Получить одно строковое значение из левой части другого.
Синтаксис:
String.left(String, Chars)
Возврат:
(С), состоящее из заданного количества символов левой стороны исходного строкового значения.
Параметры:
String - **(Л)**, исходное. **ОП**.
Chars - **(Ц)**, количество символов, взятых с левой стороны исходного строкового значения (если количество взятых символов превышает длину исходного строкового значения или равно ей, то возвращается всё исходное строковое значение целиком). **ОП**.
Пример:
```
Debug.trace(String.left("Волкодав", 4)) //результат: "Волк"
```

String.right

Получить одно строковое значение из правой части другого.

Синтаксис:

String.right(String, Chars)

Возврат:

(С), состоящее из заданного количества символов с правой стороны исходного строкового значения.

Параметры:

String - **(Л)**, исходное. **ОП**.

Chars - **(Ц)**, количество символов, взятых с правой стороны исходного строкового значения (если количество взятых символов превышает длину исходного строкового значения или равно ей, то возвращается всё исходное строковое значение целиком). **ОП**.

Пример:

```
Debug.trace(String.right("Трубкозуб", 3)) //результат: "зуб"
```

String.toupper

Преобразовать в строковом значении все символы нижнего регистра в символы верхнего регистра.

Не работает с русскими буквами.

Синтаксис:

String.toupper(String)

Возврат:

(С) со всеми символами в верхнем регистре.

Параметры:

String - **(Л)**, исходное. **ОП**.

Пример:

```
Debug.trace(String.toupper("MiNoTaUrUs")+"\n") //результат: "MINOTAURUS"
Debug.trace(String.toupper(Vector1)) //результат: "[ILMOBJECT VECTOR1]"
```

String.tolower

Преобразовать в строковом значении все символы верхнего регистра в символы нижнего регистра.

Не работает с русскими буквами.

Синтаксис:

String.tolower(String)

Возврат:

(С) со всеми символами в нижнем регистре.

Параметры:

String - **(Л)**, исходное. **ОП**.

Пример:

```
Debug.trace(String.tolower("MiNoTaUrUs")+"\n") //результат: "minotaurus"
Debug.trace(String.tolower(Vector1)) //результат: "[ilmobject vector1]"
```

String.contains

Проверить, содержит ли текст одного строкового значения другое строковое значение.

Синтаксис:

String.contains(String, Term, IgnoreCase)

Возврат:

(Б), наличие искомого строкового значения в исходном (**true** - содержится, **false** - нет).

Параметры:

String - **(Л)**, исходное, сразу преобразуется в строковое значение, в котором будет производится поиск. **ОП**.

Term - **(С)**, искомое в исходном строковом значении. **ОП**.

IgnoreCase - **(Б)**, игнорирование регистра (**true** - поиск строкового значения будет происходить без учета регистра, **false** - с учетом). **НП**, по умолчанию **false**. Игнорирование регистра **не работает с русскими буквами.**

Примеры:

С учетом регистра:

```
Phrase1="Labyrinth of the Minotaur";Searchword1="mInOtAuR"
Debug.trace(String.contains(Phrase1, Searchword1)) //результат: false
```

С игнорированием регистра

```
Phrase1="Labyrinth of the Minotaur";Searchword1="mInOtAuR"
Debug.trace(String.contains(Phrase1, Searchword1, true)) //результат: true
```

С русскими буквами:

```
Debug.trace(String.contains("Лабиринт минотавра", "минотавр")) //результат: true
Debug.trace(String.contains("Лабиринт минотавра", "кентавр")) //результат: false
```

String.word

Получить слово под указанным номером из текста строкового значения.

Синтаксис:

String.word(String, Index, Separator)

Возврат:

(С), слово из текста строкового значения. Если указан номер слова больший чем количество слов в строковом значении, то возвращается пустое строковое значение.

Параметры:

String - **(Л)**, исходное, сразу преобразуется в строковое значение, из которого будет браться слово. **ОП**.

Index - **(Ц)**, порядковый номер (нумерация начинается с **1**) нужного слова в исходном строковом значении. **ОП**.

Separator - **(С)**, разделитель между словами в исходном тексте. **НП**, по умолчанию разделителем между словами является одиночный пробел.

Примеры:

```
LongText="Жираф#@Носорог#@Бегемот#@Улитка#@Пингвин"
Debug.trace(String.word(LongText, 4, "#@")) //результат: "Улитка"
```

String.fromCharCode

Получить символ по его **ASCII**-коду.

Синтаксис:

String.fromCharCode(CharNumber)

Возврат:

(С), символ.

Параметры:

CharNumber - **(Ц)** от **0** до **255**, **ASCII**-код символа по кодировке **ISO-Latin-1**. **ОП**.

Пример:

```
for (n=0; n!=256; n++)
{Debug.trace("Код: "+n+" символ: "+String.fromCharCode(n)+"\n")
wait(0.5)}
```

String.random

Получить случайное число из заданного интервала как строковое значение.

Синтаксис:

String.random(Limit)

Возврат:

(С), случайное целое число от **0** до **N=Limit-1** включительно.

Параметры:

Limit - **(Ц)**, верхняя граница интервала из которого будет браться случайное число. **ОП**.

Пример:

```
Debug.trace(1+String.integer(String.random(99))) //результат: случайное число от 1 до 99
Debug.trace("\n"+String.random(2)) //результат: либо 0, либо 1
Debug.trace("\n"+(40+String.integer(String.random(11)))) //результат: случайное число от 40 до 50
```

String.format

Получить строковое значение из числового в нужном формате.

Синтаксис:

String.format(Format, Number)

Возврат:

(С) из числового в заданном формате.

Параметры:

Format - **(С)**, описывающее формат для числового, имеющее вид **0N.M** (**0** - необязательный символ, если указан, то перед целой частью числа будет стоять некоторое количество нулей, если нет, то пробелов; **N** - общее количество символов финального строкового значения (включая нули/пробелы впереди и точку между дробной и целой частью); **M** - количество знаков в дробной части). **ОП**.

Number - **(Ч)**, которое требуется отформатировать. **ОП**.

Примеры:

```
Debug.trace(String.format("09.3", 1.1)) //результат: "00001.100"
```

String.bool

Преобразовать любое значение или выражение в логическое значение.

Синтаксис:

String.bool(Value)

Возврат:

(Б), **true** если строковое значение содержит символы, **false** если строковое значение пустое.

(Б), **true** если выражение истинно, **false** если выражение ложно.

Параметры:

Value - **(Л)** для преобразования в логическое значение. **ОП**.

Пример:

```
Debug.trace("Строка="+String.bool(" ")+"\t"+"Пустота="+String.bool("")) //в булево значение
Debug.trace("\n"+"1="+String.bool(1)+"\t"+"0="+ String.bool(0))
Debug.trace("\n"+"Opus=Opus="+String.bool("Opus"=="Opus")+"(булево)")
```

String.integer

Преобразовать любое значение или выражение в целое число.

Синтаксис:

String.integer(Value)

Возврат:

(Ц), полученное из любого значения или выражения. Если строковое значение начинается с циферных знаков, то весь начальный циферный фрагмент до первого нециферного символа преобразится в целое число, остальная же часть строкового значения, даже содержащая цифры дальше, проигнорируется. Строковое значение, начинающееся не с циферного знака, преобразуется в **0**. Булево значение **true** (без кавычек) преобразуется в **1**, а **false** в **0**. От числового значения возвращается его целая часть.

Параметры:

Value - **(Л)** для преобразования в целое число. **ОП**.

Пример:

```
Debug.trace("\n"+"32.5 зуба="+String.integer("32.5 зуба")+" F19="+String.integer("F19"))
Debug.trace("\n"+"58.85="+String.integer(58.85)+" true="+String.integer(true))
```

String.number

Преобразовать любое значение или выражение в числовое значение.

Синтаксис:

```
String.number(Value)
```

Возврат:

(Ч), полученное из любого значения или выражения.

Параметры:

Value - **(Л)** для преобразования в числовое значение. **ОП**.

Пример:

```
Debug.trace("\n"+"32.5 зуба="+String.number("32.5 зуба")+" F19="+String.number("F19")) //в число
Debug.trace("\n"+"'2.5'+3.6+true="+(String.number("2.5")+3.6+String.number(true)))
Debug.trace("\n"+"Opus=Opus="+String.number("Opus"=="Opus")+"(числовое)")
```

String.string

Преобразовать любое значение или выражение в строковое значение.

Синтаксис:

```
String.string(Value)
```

Возврат:

(С), полученное из любого значения или выражения.

Параметры:

Value - **(Л)** для преобразования в строковое значение. **ОП**.

Пример:

```
Debug.trace("\n"+"'3.8'+5.2+false="+(String.string("3.8")+5.2+String.string(false))) //в строковое
Debug.trace("\n"+"Opus=Basic="+String.string("Opus"=="Basic")+"(строковое)")
```

escape и unescape

Закодировать (**escape**) или раскодировать (**unescape**) специальные символы в строковом значении.
Некодируемые спецсимволы: [+], [-], [*], [/], [@], [_].

Синтаксис:

```
escape(String)
unescape(String)
```

Возврат:

(С), с закодированными/раскодированными специальными символами.

Параметры:

String - **(С)** для преобразования. **ОП**.

Пример:

```
var Phrase1="ЙКЩЪПЭБЮ~#$%^&()={}[]:;|<>?.,"
var Phrase2=escape(Phrase1)
Debug.trace("Преобразование специальных символов: "+Phrase2+"\n")
Debug.trace("Обратное преобразование: "+unescape(Phrase2))
```

CountLines

Получить количество строчек в указанном строковом значении, графическом текстовом объекте или текстовом файле. Не работает напрямую со строковым объектом **String**, его требуется преобразовать в строковое значение, либо функцией **string** глобального строкового объекта, либо функцией **toString**.
ВАЖНО: Для графических текстовых объектов лучше использовать, выполняющую эту же задачу, специализированную функцию **GetLineCount**.

Синтаксис:

```
CountLines(Name)
```

Возврат:

(Ц), количество строчек в указанном строковом значении, текстовом объекте или текстовом файле.

Параметры:

Name - **(С)**, либо **(О)** (текстовый объект или открытый текстовый файл). **ОП**.

Примеры:

Подсчёт количества строчек в строковом значении:

```
var Phrase1="Олень\nЛось\nКабан\nВолк"
Debug.trace(CountLines(Phrase1)+"\n"+CountLines("Лиса\nЗаяц\nБарсук\nМедведь"))
```

Подсчёт количества строчек в строковом объекте:

```
var Phrase2=new String("Черепаха\nЯщерица\nДинозавр")
Debug.trace(CountLines(String.string(Phrase2)))
Debug.trace("\n"+CountLines(Phrase2.toString()))
```

Подсчёт количества строчек в текстовом файле:

```
var Phrase3=OpenFile(SYSTEM_PUBLICATION_DIR+"\\DATA\\TestText.txt") //открытие файла
Debug.trace(CountLines(Phrase3.Read()))
```

Подсчёт количества строчек в текстовом объекте:

```
Debug.trace(CountLines(Text1)) //Text1 - графический текстовый объект, созданный в органайзере
```

CountWords

Получить количество слов в указанном строковом значении, строковом объекте, графическом текстовом объекте или текстовом файле. Не работает напрямую со строковым объектом **String**, его требуется сконвертировать в строковое значение, либо функцией **string** глобального строкового объекта, либо функцией **toString**.
ВАЖНО: Для графических текстовых объектов лучше использовать, выполняющую эту же задачу, специализированную функцию **GetWordCount**.

Синтаксис:

```
CountWords(Name)
```

Возврат:

(Ц), количество слов в указанном строковом значении, текстовом объекте или текстовом файле.

Параметры:

Name - **(С)**, либо **(О)** (текстовый объект или открытый текстовый файл). **ОП**.

Примеры:

Подсчёт количества слов в строковом значении:

```
var Phrase1="Весело играют в сказочном саду"
Debug.trace(CountWords(Phrase1)+"\n"+CountWords("дино динозавры в сладкую игру"))
```

Подсчёт количества слов в строковом объекте:

```
var Phrase2=new String("Кто бежит быстрее по тропинкам сада")
var Phrase3=new String("тот получит первым приз из шоколада")
Debug.trace(CountWords(String.string(Phrase2)))
Debug.trace("\n"+CountWords(Phrase3.toString()))
```

Подсчёт количества слов в текстовом файле:

```
var Phrase4=OpenFile(SYSTEM_PUBLICATION_DIR+"\\DATA\\TestText.txt") //открытие файла
Debug.trace(CountWords(Phrase4.Read()))
```

Подсчёт количества слов в текстовом объекте:

```
Debug.trace(CountWords(Text1)) //Text1 - графический текстовый объект, созданный в органайзере
```

StringReplace

Преобразовать любой тип данных в строковое значение с заменой в содержании всех повторяющихся фрагментов текста на указанное значение. У объектов, созданных в органайзере, преобразуется их название в строковое значение, а у объектов, созданных скриптом, преобразуется их значение в строковое.

Синтаксис:

StringReplace(MainValue, OldValue, NewValue)

Возврат:

(С), полученное от преобразования указанных данных.

Параметры:

MainValue - **(Л)**, сразу преобразуется в строковое значение, основное строковое значение. **ОП**.

OldValue - **(Л)**, сразу преобразуется в строковое значение, фрагмент текста, чьи повторения в содержании основного строкового значения, будут заменяться новым значением. **ОП**.

NewValue - **(Л)**, сразу преобразуется в строковое значение, новое строковое значение для замены указанных повторяющихся фрагментов текста в основном строковом значении. **ОП**.

Пример:

```
//Замена всех слов Black на White в строковом значении
var Phrase1="My Black Bicycle, My Black Bicycle"
Debug.trace(StringReplace(Phrase1, "Black", "White")+"\n")
//Замена всех слов Green на Yellow в строковом значении строкового объекта
var Phrase2=new String("Green Submarine, Green Submarine")
Debug.trace(StringReplace(Phrase2, "Green", "Yellow")+"\n")
//Замена всех цифр 1 на цифру 2 в числовом значении и пребразование в строковое
var Number1=111.111;Debug.trace(StringReplace(Number1, 1, 2)+3+"\n")
//Из названия текстового объекта получено строковое значение "[ILMObject Text1]"
Debug.trace(StringReplace(Text1, 1, 8)) //результат: "[ILMObject Text8]"
```

ToClipboard

Скопировать в буфер обмена любой тип данных.

Синтаксис:

ToClipboard(Value, Native)

Параметры:

Value - **(Л)**, для копирования в буфер обмена.

Native - **(Б)**, тип копируемых данных только для текстовых графических объектов (**true** - текст с форматированием, **false** - текст как изображение). **НП**, по умолчанию **false**.

Пример:

```
var Phrase1="Слон, носорог, бегемот";ToClipboard(Phrase1, false)
```

PrintString

Распечатать указанное строковое значение со шрифтом по умолчанию.

Синтаксис:

PrintString(String, PrintDialog, CancelDialog, Landscape)

Параметры:

String - **(С)**, для печати.

PrintDialog - **(Б)**, отображение окна свойств печати перед распечаткой (**true** - окно появится, **false** - нет). **НП**, по умолчанию **false**.

CancelDialog - **(Б)**, отображение окна отмены печати во время распечатки (**true** - окно появится, **false** - нет). **НП**, по умолчанию **true**.

Landscape - **(Б)**, ориентация печати (**true** - альбомная, **false** - портретная). **НП**, по умолчанию **true**.

Пример:

```
PrintString("Проверка печати", false, false, false)
```

GenerateGUID

Сгенерировать глобально-уникальный идентификатор (**Globally Unique Identifier**).

Синтаксис:

GenerateGUID()

Возврат:

(С), **ГУИД**, имеющий вид **"XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"**, где **X** - это цифры или буквы **A-F**.

Пример:

```
Debug.trace("ГУИД: "+GenerateGUID())
```

length

Определить количество символов в строковом значении.

Синтаксис:

length

Возврат:

(Ц), количество символов (включая пробелы) в строковом значении.

Пример:

```
var Word1="<Лабиринт минотавра>"
Debug.trace(Word1.length) //результат: 20
```

charAt

Получить символ строкового значения под указанным номером.

Синтаксис:

charAt(index)

Возврат:

(С), символ строкового значения.

Параметры:

index - **(Ц)**, порядковый номер символа в строковом значении (нумерация начинается с **0**). **НП**, по умолчанию **0**.

Пример:

```
var Word1="Вампир"
Debug.trace(Word1.charAt(2)) //третий символ строки, результат: "м"
```

charCodeAt

Получить **ASCII**-код указанного символа в строковом значении.

Синтаксис:

charCodeAt(index)

Возврат:

(Ц), **ASCII**-код указанного символа строкового значения по кодировке **ISO-Latin-1**.

Параметры:

index - **(Ц)**, порядковый номер символа в строковом значении (нумерация начинается с **0**). **НП**, по умолчанию **0**.

Пример:

```
var Word1="Dungeons & Dragons"
Debug.trace (Word1.charCodeAt(9)) //код символа "&", результат: 38
```

toUpperCase

Преобразовать в строковом значении все символы нижнего регистра в символы верхнего регистра.

Не работает с русскими буквами.

Синтаксис:

toUpperCase()

Возврат:

(С) со всеми символами в верхнем регистре.

Пример:

```
var Word1="MiNoTaUrUs"
Debug.trace(Word1.toUpperCase()) //результат: "MINOTAURUS"
```

toLowerCase

Преобразовать в строковом значении все символы верхнего регистра в символы нижнего регистра.
Не работает с русскими буквами.
Синтаксис:
toLowerCase()
Возврат:
(С) со всеми символами в нижнем регистре.
Пример:

```
var Word1="MiNoTaUrUs"
Debug.trace(Word1.toLowerCase()) //результат: "minotaurus"
```

substring

Получить указанную часть строкового значения.
Синтаксис:
substring(From, To)
Возврат:
(С), указанная часть строкового значения.
Параметры:
From - **(Ц)**, порядковый номер (нумерация начинается с **0**) символа в исходном строковом значении, с которого начнется новое строковое значение. **ОП**.
To - **(Ц)**, порядковый номер (нумерация начинается с **0**) символа в исходном строковом значении, до которого (сам символ не входит) берется новое строковое значение. **НП**, по умолчанию **-1** (до конца исходного строкового значения, включая последний символ).
Пример:

```
var Phrase1="Лабиринт минотавра"
Debug.trace(Phrase1.substring(9, 17)) //результат: "минотавр"
```

split

Разбить строковое значение на части, каждая из которых станет элементом массива.
Синтаксис:
split(separator, limit)
Возврат:
(О), массив из частей строкового значения.
Параметры:
separator - **(С)**, обозначение разделителя. **НП**, по умолчанию всё строковое значение будет единственным элементом массива.
limit - **(Ц)**, ограничение на количество разбиений. **НП**. **Параметр не работает**, ограничение игнорируется.
Пример:

```
var LongPhrase="Жираф#@Носорог#@Бегемот#@Улитка#@Пингвин" //исходное строковое значение
var Array1=LongPhrase.split("#@", 2) //разделитель #@, ограничение на два разбиения не работает
for (m=0; m<5; m++) {wait(0.5);Debug.trace(Array1[m]+"\n")} //вывод значений пяти элементов массива
```

indexOf и lastIndexOf

Найти строковое значение с указанной позиции внутри другого строкового значения. При использовании **indexOf** поиск ведется слева направо, а при использовании **lastIndexOf** справа налево.

Синтаксис:

```
indexOf(substring, index)
lastIndexOf(substring, index)
```

Возврат:

(Ц), положение символа в исходном строковом значении, с которого начинается найденное искомое строковое значение. Если искомое строковое значение не найдено, то **-1**.

Параметры:

substring - **(С)**, искомое. **ОП**.

index - **(Ц)**, позиция (порядковый номер, начинающийся с **0**) символа исходного строкового значения, с которого ведется поиск (включая его). **НП**, по умолчанию **0**.

Пример:

```
var Phrase1=new String("Один слон ест бананы, а другой слон ест ананасы")
Debug.trace("С самого начала: "+Phrase1.indexOf("слон")) //5, первое слово "слон"
Debug.trace("\n"+"С самого конца: "+Phrase1.lastIndexOf("слон")) //31, второе слово "слон"
Debug.trace("\n"+"С 6 символа направо: "+Phrase1.indexOf("слон", 6)) //31, второе слово "слон"
Debug.trace("\n"+"С 30 символа налево: "+Phrase1.lastIndexOf("слон", 30)) //5, первое слово "слон"
```

Работа с файлами

Файловые функции предназначены для открытия, чтения и изменения файлов. Чтобы использовать эти функции, необходимо сначала создать новый файловый объект в скрипте с помощью функции **OpenFile**. При использовании файловых функций названия файловых путей прописываются не с одиночным обратным слешем, например как **"C:\GAMES\README.TXT"**, а с двойным обратным слешем, например: **"C:\\GAMES\\README.TXT"**. Для запуска файлов из места, где расположен файл публикации, используется **SYSTEM_PUBLICATION_DIR**, системная переменная. Файловые пути также могут задаваться заранее как псевдонимы (**alias**) в самой программе **Opus Pro** (пункт меню **"Publication"** > команда **"Publication Properties"** > вкладка **"Additional Resources"**).

Список функций:

FileExists - проверить наличие файла.
GetFileVersion - получить версию **EXE** или **DLL**-файла.
LaunchFile - открыть внешний файл.
CopyFile - скопировать файл.
DeleteFile - удалить файл.
PrintFile - распечатать файл.
OpenFile - открыть/создать текстовый файл.
Read - прочитать весь текст в открытом файле.
ReadLine - прочитать строчку в открытом файле.
ReadField - прочитать фрагмент текста в открытом файле.
Write - записать данные в открытый файл.
WriteLine - записать данные на следующей строчке в открытом файле.
WriteField - записать фрагмент данных в открытом файле.
Close - закрыть открытый файл.
GotoFirstLine - перейти к первой строчке открытого файла.
GotoNextLine - перейти к следующей строчке открытого файла.
EndOfFile - проверить достигнут ли конец открытого файла.
WinHelp - открыть **HLP**-файл справки.
HtmlHelp - открыть **CHM**-файл справки.
GetINIFileData - получить значение конкретного ключа в заданном разделе **INI**-файла.
GetINISectionData - получить значения всех ключей заданного раздела **INI**-файла.
SetIniFiledata - установить значение конкретного ключа в заданном разделе **INI**-файла.
ReportSetFilename - открыть/создать **LOG**-файл.
ReportGetFilename - получить путь к уже открытому **LOG**-файлу.
ReportWriteString - записать данные в уже открытый **LOG**-файл.
ReportClose - закрыть уже открытый **LOG**-файл.

Для проверки примеров к файловым функциям необходимо, в папке где сохранена тестовая публикация, создать папку **DATA** и поместить туда текстовый файл **TestText.txt** с содержимым:

12345
"XYZWV", "РЫБА", "ФРУКТ"
67890
Жираф
Верблюд
Динозавр

FileExists

Проверить наличие файла по указанному пути.

Синтаксис:

```
FileExists(Filename)
```

Возврат:

(Ц), показатель наличия файла, возможные значения:
0 - файла по указанному пути не существует.
1 - файл по указанному пути существует.
2 - файл по указанному пути существует, но имеет атрибут **"Только чтение"**.

Параметры:

Filename - **(С)**, полный путь к проверяемому файлу. **ОП**.

Пример:

```
var FileCheck=FileExists(SYSTEM_PUBLICATION_DIR+"\\Readme.txt") //проверка наличия файла
if (FileCheck==0) {Debug.trace("Файла не существует")} //вывод результата если такого файла нет
else {Debug.trace("Файл существует")} //вывод результата если такой файл есть
```

GetFileVersion

Получить версию указанного **EXE** или **DLL**-файла.

Синтаксис:

```
GetFileVersion(Filename)
```

Возврат:

(С), версия проверяемого файла.

Параметры:

Filename - **(С)**, полный путь к проверяемому файлу. **ОП**.

Пример:

```
Debug.trace(GetFileVersion(SYSTEM_PROGRAMS_DIR+"\\Opus Pro 9\\OpusPro.exe")+"\n")
Debug.trace(GetFileVersion(SYSTEM_PROGRAMS_DIR+"\\Opus Pro 9\\lame_enc.dll"))
```

LaunchFile

Запустить указанную программу или файл, который запустится в ассоциированной с ним программе.

Синтаксис:

```
LaunchFile(Filename, Parameters)
```

Параметры:

Filename - **(С)**, путь к запускаемому файлу. **ОП**.
Parameters - **(С)**, параметры командной строки. **НП**.

Пример:

```
//Открыть файл в связанной с ним программе
LaunchFile(SYSTEM_PUBLICATION_DIR+"\\Readme.txt")
//Запустить программу, а в ней открыть текстовый файл из параметров
var File1=SYSTEM_WIN_DIR+"\\notepad.exe"
var Params=SYSTEM_PUBLICATION_DIR+"\\Readme.txt"
LaunchFile(File1, Params)
```

CopyFile

Скопировать на компьютере указанный файл. Директории копировать нельзя.

Синтаксис:

CopyFile(Source, Destination, MsgBox)

Возврат:

(Б), результат копирования (**true** - процесс прошел успешно, **false** - нет).

Параметры:

Source - **(С)**, путь к копируемому файлу. **ОП**.

Destination - **(С)**, путь, куда следует скопировать файл. Если указанных в пути директорий не существует, то они создадутся автоматически. **ОП**, может быть введен следующими способами:

название файла без указания пути - копия файла с указанным названием будет располагаться там же, где находится копируемый файл.

полный путь без указания названия файла - копия файла с тем же названием, что у исходного копируемого файла будет располагаться по указанному пути. Важно завершить такой путь символами [\\ ], иначе последняя директория будет считаться названием файла без расширения.

полный путь с указанием названия файла - копия файла с указанным названием будет располагаться по указанному пути.

MsgBox - **(Б)**, отображение окна предупреждения о замене файла с совпадающим названием (**true** - окно появится, **false** - нет). **НП**, по умолчанию **false**.

Пример:

```
//Копирование файла в ту же папку
CopyFile(SYSTEM_PUBLICATION_DIR+"\\Readme.txt", "Прочти.txt")
```

DeleteFile

Удалить с компьютера указанный файл или директорию.

Синтаксис:

DeleteFile(Filename, MsgBox, Recycle)

Возврат:

(Б), результат удаления (**true** - процесс прошел успешно, **false** - нет).

Параметры:

Filename - **(С)**, полный путь к файлу/директории. **ОП**.

MsgBox - **(Б)**, отображение окна предупреждения об удалении (**true** - окно появится, **false** - нет). **НП**, по умолчанию **false**.

Recycle - **(Б)**, отправка файла/директории в корзину (**true** - отправка в корзину, **false** - безвозвратное удаление). **НП**, по умолчанию **false**.

Пример:

```
DeleteFile("C:\\Games\\Readme.txt", true, true)
```

PrintFile

Распечатать указанный файл.

Синтаксис:

PrintFile(Filename, PrintDialog, CancelDialog)

Параметры:

Filename - **(С)**, название файла для печати. **ОП**.

PrintDialog - **(Б)**, отображение окна свойств печати перед распечаткой (**true** - окно появится, **false** - нет). **НП**, по умолчанию **false**.

CancelDialog - **(Б)**, отображение окна отмены печати во время распечатки (**true** - окно появится, **false** - нет). **НП**, по умолчанию **true**.

Пример:

```
PrintFile(SYSTEM_PUBLICATION_DIR+"\\Readme.txt", false, false)
```

OpenFile

Открыть для дальнейших операций уже созданный текстовый файл. Если такого файла не существует, то по указанному пути создается новый пустой текстовый файл (вместе с директориями, которых недостает).

Синтаксис:

OpenFile(Filename, Overwrite, ReadOnly)

Возврат:

(О), текстовый файл.

Параметры:

Filename - **(С)**, полный файловый путь к указанному текстовому файлу. **ОП**. Если указанного файла не существует, то автоматически создается новый файл.

Overwrite - **(Б)**, перезапись файла (**true** - полностью перезаписывается, **false** - новые данные записываются в конец файла). **НП**, по умолчанию **false**.

ReadOnly - **(Б)**, установка режима **"Только чтение"** (**true** - режим включен и в файл нельзя вносить изменения (в этом случае параметр **Overwrite** не может быть **true**), **false** - режим выключен и файл может быть изменен). **НП**, по умолчанию **false**.

Read

Прочитать файл, открытый/созданный функцией **OpenFile**.

Синтаксис:

Read(Encrypt, Key)

Возврат:

(С), всё содержимое файла.

Параметры:

Encrypt - **(Б)**, чтение файла (**true** - с расшифровкой, **false** - как есть). **НП**, по умолчанию **false**.

Key - **(С)**, ключ шифрования, если есть. **НП**.

ReadLine

Прочитать строчку файла (начиная с первой), открытого/созданного функцией **OpenFile**. После каждого повторного вызова функции будет читаться следующая строчка.

Синтаксис:

ReadLine(Encrypt, Key)

Возврат:

(С), строчка файла.

Параметры:

Encrypt - **(Б)**, чтение файла (**true** - с расшифровкой, **false** - как есть). **НП**, по умолчанию **false**.

Key - **(С)**, ключ шифрования, если есть. **НП**.

ReadField

Прочитать фрагмент файла до запятой, открытого/созданного функцией **OpenFile**. После каждого повторного вызова функции будет читаться следующий фрагмент.

Синтаксис:

ReadField(Quote, Encrypt, Key)

Возврат:

(С), фрагмент текста до запятой. Если текст не разделен запятыми, то возвращается полный текст.

Параметры:

Quote - **(Б)**, проверка кавычек (**true** - для фрагментов, заключенных в кавычки, **false** - для фрагментов не заключенных в кавычки). **НП**, по умолчанию **true**.

Encrypt - **(Б)**, чтение файла (**true** - с расшифровкой, **false** - как есть). **НП**, по умолчанию **false**.

Key - **(С)**, ключ шифрования, если есть. **НП**.

Общий пример:

```
var Data1=OpenFile(SYSTEM_PUBLICATION_DIR+"DATA\\TestText.txt") //открытие файла
Debug.trace(Data1.Read()) //чтение всего текста
Debug.trace(Data1.ReadLine()) //чтение строчки
Debug.trace(Data1.ReadField()) //чтение фрагмента
```

Write

Записать строковое значение в конец последней существующей строчки файла, открытого/созданного функцией **OpenFile**.

Синтаксис:

Write(String, Encrypt, Key)

Параметры:

String - **(С)**, записываемое в файл. **ОП**.

Encrypt - **(Б)**, шифрование (**true** - файл зашифровывается, **false** - не зашифровывется). **НП**, по умолчанию **false**.

Key - **(С)**, ключ шифрования. **НП**.

WriteLine

Записать строковое значение на новой строчке в конце файла, открытого/созданного функцией **OpenFile**.
Если перед этой функцией производилась запись данных в файл с помощью **Write** или **Writefield**, то перехода на следующую строчку не будет, а текущие данные запишутся в конец последней строчки.

Синтаксис:

WriteLine(String, Quote, Encrypt, Key)

Параметры:

String - **(С)**, записываемое в файл. **ОП**.

Quote - **(Б)**, добавление кавычек к строковому значению, записываемому в файл (**true** - заключить в кавычки, **false** - оставить без кавычек). **НП**, по умолчанию **false**.

Encrypt - **(Б)**, шифрование (**true** - файл зашифровывается, **false** - не зашифровывется). **НП**, по умолчанию **false**.

Key - **(С)**, ключ шифрования. **НП**.

WriteField

Записать через запятую строковое значение в конец последней существующей строчки файла, открытого/созданного функцией **OpenFile**.
Если перед этой функцией производилась запись данных в файл с помощью **Write** или **Writeline**, то произойдет переход на следующую строчку, а текущие данные запишутся без запятой в начале.

Синтаксис:

WriteField(String, Quote, Encrypt, Key)

Параметры:

String - **(С)**, записываемое в файл. **ОП**.

Quote - **(Б)**, добавление кавычек к строковому значению, записываемому в файл (**true** - заключить в кавычки, **false** - оставить без кавычек). **НП**, по умолчанию **true**.

Encrypt - **(Б)**, шифрование (**true** - файл зашифровывается, **false** - не зашифровывется). **НП**, по умолчанию **false**.

Key - **(С)**, ключом шифрования. **НП**.

Close

Закрыть файл, открытый/созданный функцией **OpenFile**.

Синтаксис:

Close()

Общий пример:

```
var Data1=OpenFile(SYSTEM_PUBLICATION_DIR+"DATA\\TestText.txt") //открытие файла
Data1.Write("Звериный Теночтитлан") //запись в файл
Data1.WriteLine("Тропа тапиров") //запись в файл
Data1.WriteField("Заросли ягуаров") //запись в файл
Debug.trace(Data1.Read()) //чтение всего текста из файла
Data1.Close() //закрытие файла
```

GotoFirstLine

Перейти к первой строчке файла, открытого/созданного функцией **OpenFile**.

Синтаксис:

GotoFirstLine()

GotoNextLine

Перейти к следующей строчке файла, открытого/созданного функцией **OpenFile**.

Синтаксис:

GotoNextLine()

EndOfFile

Проверить достигнут ли конец файла, открытого/созданного функцией **OpenFile**.

Синтаксис:

EndOfFile()

Возврат:

(Б), результат проверки (**true** - конец файла достигнут, **false** - нет).

Общий пример:

```
var Data1=OpenFile(SYSTEM_PUBLICATION_DIR+"DATA\\TestText.txt") //открытие файла
n=0;Data1.GotoFirstLine() //переход к первой строчке файла
//переход к следующей строчке пока не достигнут конец файла
while (Data1.EndOfFile()==false) {wait(0.1);n=n+1;Data1.GotoNextLine()}
Debug.trace("Количество строчек: "+n)
```

HtmlHelp

Открыть **CHM**-файл справки в указанном разделе (требуется версия **Windows** старше **2000**, с **Internet Explorer 4** или выше).

Синтаксис:

HtmlHelp(Filename, ID)

Параметры:

Filename - **(С)**, путь к **CHM**-файлу справки. **ОП**.

ID - **(Ц)**, номер раздела справки для перехода. **НП**, по умолчанию **0**.

Пример:

```
HtmlHelp(SYSTEM_PROGRAMS_DIR+"Opus Pro 9\\OpusScript.chm", 32)
```

WinHelp

Открыть **HLP**-файл справки в указанном разделе (не работает в последних версиях **Windows**).

Синтаксис:

WinHelp(Filename, ID)

Параметры:

Filename - **(С)**, путь к **HLP**-файлу справки. **ОП**.

ID - **(Ц)**, номер раздела справки для перехода. **НП**, по умолчанию **0**.

GetINIFileData и GetINISectionData

Получить значения из указанного **INI**-файла.

Синтаксис:

GetINIFileData(FilePath, Section, Key)
GetINISectionData(FilePath, Section)

Возврат:

GetINIFileData возвращает:
(C), значение указанного ключа если такой существует, иначе пустая строка.
GetINISectionData возвращает:
(C), список всех ключей указанного раздела, вида **"Название ключа=Значение ключа"**, где сведения о каждом ключе будет располагаться на новой строчке.

Параметры:

FilePath - **(C)**, путь к **INI**-файлу. **ОП**.
Section - **(C)**, название раздела (без квадратных скобок) в **INI**-файле. **ОП**.
Key - **(C)**, название ключа раздела. **ОП**.

Пример:

```
var Path1="Opus Pro 9\\Samples\\Calculator\\BasicCalc_Info.ini"
var KV1=GetINIFileData(SYSTEM_PROGRAMS_DIR+Path1, "Sample", "Desc")
var SV1=GetINISectionData(SYSTEM_PROGRAMS_DIR+Path1, "Sample")
Debug.trace("Значение ключа:\n"+KV1+"\n"+"Данные раздела:\n"+SV1)
```

SetINIFileData

Установить значение для определенного ключа нужного раздела в указанном **INI**-файле.

Синтаксис:

SetINIFileData(FilePath, Section, Key, Value)

Возврат:

(Б), результат записи данных (**true** - успешный, **false** - произошла ошибка).

Параметры:

FilePath - **(C)**, путь к **INI**-файлу. **ОП**.
Section - **(C)**, название раздела (без квадратных скобок) в **INI**-файле. **ОП**.
Key - **(C)**, название ключа раздела. **ОП**.
Value - **(C, Ц)**, значение для установки. **ОП**.

Пример:

```
var Path1="Opus Pro 9\\Samples\\Calculator\\BasicCalc_Info.ini"
var KV1="Обычный калькулятор"
SetINIFileData(SYSTEM_PROGRAMS_DIR+Path1, "Sample", "Desc", KV1)
```

ReportSetFilename

Открыть/создать **LOG**-файл и назначить его для ведения отчета. Разрешается вести не более пяти **LOG** -файлов одновременно.

Синтаксис:

ReportSetFilename(Filename, Append, ReportNumber)

Возврат:

(Б), успешность открытия **LOG**-файла (**true** - файл открылся, **false** - произошла ошибка).

Параметры:

Filename - **(С)**, путь к файлу с расширением **.LOG** (если расширение не указывать, то оно присвоится автоматически). **ОП**.

Append - **(Б)**, режим записи данных (**true** - добавление новых к существующим, **false** - замена существующих на новые). **НП**, по умолчанию **true**.

ReportNumber - **(Ц)** от **0** до **4**, номер присваиваемый **LOG**-файлу. **НП**, по умолчанию **0**.

Пример:

```
ReportSetFilename(SYSTEM_PUBLICATION_DIR+"Journal.log", true, 1)
```

ReportGetFilename

Получить название (с полным путем) уже открытого **LOG**-файла по его номеру.

Синтаксис:

ReportGetFilename(ReportNumber)

Возврат:

(С), полный путь к **LOG**-файлу. Если **LOG**-файл не открыт, то пустое строковое значение.

Параметры:

ReportNumber - **(Ц)** от **0** до **4**, номер открытого **LOG**-файла. **НП**, по умолчанию **0**.

Пример:

```
Debug.trace("Название LOG-файла: "+ReportGetFilename(1))
```

ReportWriteString

Записать данные в уже открытый **LOG**-файл (если файл не окрыт, то ничего не произойдет).

Синтаксис:

ReportWriteString(Text, ReportNumber)

Параметры:

Text - **(С)**, текст для записи в **LOG**-файл. **ОП**.

ReportNumber - **(Ц)** от **0** до **4**, номер открытого **LOG**-файла. **НП**, по умолчанию **0**.

Пример:

```
ReportWriteString("Гильгамеш", 1)
```

ReportClose

Закрыть уже открытый **LOG**-файл.

Синтаксис:

ReportClose(ReportNumber)

Параметры:

ReportNumber - **(Ц)** от **0** до **4**, номер открытого **LOG**-файла, **-1** для закрытия всех открытых **LOG**-файлов. **НП**, по умолчанию **0**.

Пример:

```
ReportClose(-1)
```

Клавиатура, мышь и джойстик

Любой объект в публикации, к которому могут быть добавлены действия, может использовать функции событий. Эти функции позволяют создавать триггер из скрипта, а не использовать триггеры в диалоге действий объекта. Объекты браузера являются единственным типом объектов, которые не могут использовать функции событий.

Список функций:

GetJoystickState - получить текущее состояние первого системного джойстика.
TriggerComponentEvent - активировать пользовательский триггер.
RegisterEventHandler - зарегистрировать событие клавиатуры или мыши.
UnregisterEventHandler - удалить уже зарегистрированное событие клавиатуры или мыши.
IsKeyPressed - проверить, нажата ли конкретная клавиша.
GetKeyState - получить текущее состояние **Lock**-клавиш.
SetKeyState - изменить состояние **Lock**-клавиш.
GetMousePosition - получить координаты текущей позиции мыши.
IsMousePressed - проверить, нажата ли указанная кнопка мыши.
MouseMove - переместить курсор мыши.
MouseClick - сымитировать нажатие кнопки мыши (возможно **нерабочая** функция).

GetJoystickState

Получить текущее состояние первого доступного джойстика.

Синтаксис:

GetJoystickState()

Возврат:

(О) со следующими свойствами:
x - **(Ч)** от **-1.0** до **1.0**, **X**-ось.
y - **(Ч)** от **-1.0** до **1.0**, **Y**-ось.
z - **(Ч)** от **-1.0** до **1.0**, **Z**-ось.
s - **(Ч)** от **-1.0** до **1.0**, слайдер.
pov - **(Ч)** от **0** до **360**, угол точки обзора в градусах, (**-1** если центрирован).
f1 - **(Ц)**, первая кнопка стрельбы, **1** если нажата, **0** если не нажата.
f2 - **(Ц)**, вторая кнопка стрельбы, **1** если нажата, **0** если не нажата.
f3 - **(Ц)**, третья кнопка стрельбы, **1** если нажата, **0** если не нажата.
f4 - **(Ц)**, четвертая кнопка стрельбы, **1** если нажата, **0** если не нажата.

Пример:

```
while (true) {wait(1)
JState=GetJoystickState()
Debug.trace("X-ось: "+JState.x+" Y-ось: "+JState.y+" Z-ось: "+JState.z+"\n")
Debug.trace("Слайдер: "+JState.s+"Точка обзора: "+JState.pov+"\n")}
```

TriggerComponentEvent

Активировать пользовательский триггер для указанного объекта.

Синтаксис:

TriggerComponentEvent(CustomTrigger)

Параметры:

CustomTrigger - **(С)**, название пользовательского триггера для активации. **ОП**.

Пример:

```
Button1.TriggerComponentEvent("Trigger1")
```

RegisterEventHandler

Зарегистрировать событие (вызов пользовательской функции, происходящий только при определенном использовании клавиатуры или мыши).

Синтаксис:

ObjectName.RegisterEventHandler(EventName, EventFunction)

Пояснение:

ObjectName - объект для которого регистрируется событие. Если не указать, то функция глобальная, а событие регистрируется для текущей страницы.

Возврат:

(Ц), идентификатор зарегистрированного события.

Параметры:

EventName - **(С)**, название события, при котором произойдёт вызов пользовательской функции. **ОП**.

События клавиатуры:

"keydown" - была нажата любая клавиша.
"keyup" - была отпущена любая нажатая клавиша.
"keypress" - была нажата клавиша, кодирующая символ.

Свойства событий клавиатуры:

key - **(Ц)**, **ASCII**-код нажатой клавиши.
autorepeat - **(Ц)**, количество повторений события.

События мыши:

"lmousedown" - была нажата левая кнопка мыши.
"lmouseup" - была отпущена левая кнопка мыши.
"lclick" - была нажата левая кнопка мыши, когда курсор находился над объектом.
"ldblclick" - была нажата дважды левая кнопка мыши, когда курсор находился над объектом.
"rmousedown" - была нажата правая кнопка мыши.
"rmouseup" - была отпущена правая кнопка мыши.
"rclick" - была нажата правая кнопка мыши, когда курсор находился над объектом.
"rdblclick" - была нажата дважды правая кнопка мыши, когда курсор находился над объектом.
"mousemove" - было произведено движение мышью.
"mouseover" - курсор мыши стал находиться над объектом.
"mouseout" - курсор мыши перестал находиться над объектом.
"mousewheelup" - было кручение колеса мыши вверх.
"mousewheeldown" - было кручение колеса мыши вниз.

Свойства событий мыши:

x - **(Ц)**, **X**-координата курсора мыши, относительно верхнего левого угла страницы.
y - **(Ц)**, **Y**-координата курсора мыши, относительно верхнего левого угла страницы.

EventFunction - название пользовательской функции (без указания кавычек, круглых скобок и параметров), вызываемой при событии. Сама же пользовательская функция может содержать только один параметр, который будет названием объекта свойств события. **ОП**.

UnregisterEventHandler

Удалить уже зарегистрированное событие.

Синтаксис:

UnregisterEventHandler(EventID)

Параметры:

EventID - **(Ц)**, идентификатор зарегистрированного события. **ОП**.

Общие примеры:

```
//1) Использование событий клавиатуры, скрипт пишется в скриптовом объекте страницы:
Event1=RegisterEventHandler("keydown", KeyChecker) //регистрация события на нажатие клавиши
function KeyChecker(KeyNumber) //объявление функции события с параметром объекта его свойств
{Debug.trace("Код нажатой клавиши: "+KeyNumber.key+"\n")}
wait(5);UnregisterEventHandler(Event1) //удаление зарегистрированного события через 5 секунд

//2) Использование событий мыши, скрипт пишется в скриптовом объекте страницы:
Event1=RegisterEventHandler("mousewheelup", Anticlockwise)
Event2=RegisterEventHandler("mousewheeldown", Clockwise)
Event3=RegisterEventHandler("mousemove", MouseChecker)
function Anticlockwise() {Vector1.Rotate(-10)}
function Clockwise() {Vector1.Rotate(10)}
function MouseChecker(MousePosition)
{Debug.trace("Координаты мыши: x="+MousePosition.x+" y="+MousePosition.y+"\n")}

//3) Скрипт полностью пишется в скриптовом объекте страницы:
Vector1.RegisterEventHandler("lclick", Twist)
Clone1=Vector1.CloneObject(200, 200, true)
function Twist()
{this.Spin(180, 3, false)
this.Roll(180, 3, false)}
//Объект Vector1 при нажатиии на него левой кнопкой мыши начнет крутиться
//Объект Clone1 при нажатии на него левой кнопкой мыши крутиться НЕ будет
//Клонам не наследуются события оригинала

//4) Основная часть скрипта пишется в скриптовом объекте страницы:
Clone1=Vector1.CloneObject(200, 200, true)
function Twist()
{this.Spin(180, 3, false)
this.Roll(180, 3, false)}
//Регистрация события пишется в скриптовом объекте, прикрепленном к дочернему элементу
страницы Vector1:
this.RegisterEventHandler("lclick", Twist)
//Объект Vector1 при нажатиии на него левой кнопкой мыши начнет крутиться
//Объект Clone1 при нажатии на него левой кнопкой мыши также начнет крутиться
//Клонам наследуются прикрепленные скриптовые объекты оригинала
```

IsKeyPressed

Получить текущее состояние нажатия клавиатурных клавиш. **ВАЖНО**: функция **НЕ проверяет комбинацию** из нажатых клавиш, а проверяет нажата ли хоть одна клавиша из указанных.

Синтаксис:

IsKeyPressed(Key1, Key2, ...)

Возврат:

(Б), состояние клавиш (**true** - хотя бы одна из указанных клавиш нажата, иначе **false**).

Параметры:

KeyN - **(Ц)**, **ASCII**-код клавиши, либо **(С)**, обозначение клавиши. **ОП**. Возможные значения:
Клавиши перемещения: **"Up"**, **"Down"**, **"Left"**, **"Right"**, **"PageUp"**, **"PageDown"**, **"End"**, **"Home"**;
Клавиши управления: **"Enter"**, **"Escape"**, **"Tab"**, **"Alt"**, **"Control"**, **"Shift"**, **"Insert"**, **"Delete"**, **"Backspace"**, **"Pause"**, **"Print"**, **"Help"**, **"LeftWin"**, **"RightWin"**, **"Caps"**, **"NumLock"**;
Клавиши цифровой клавиатуры: **"Numpad0"** - **"Numpad9"**, **"Add"** [+], **"Subtract"** [-], **"Multiply"** [*], **"Divide"** [/], **"Decimal"** [.];
Функциональные клавиши: **"F1"** - **"F24"**;
Буквенные клавиши: **"a"** - **"z"**, либо **"A"** - **"Z"** (регистр ни на что не влияет).

Пример:

```
while (true) {wait(0.1) //бесконечный цикл с защитой от перегрузки
//нажатие разных клавиш приводит к выводу различных сообщений в скриптовой консоли
if (IsKeyPressed("A")) {Debug.trace("Мяу!\n")} //Мяу! если нажата A
if (IsKeyPressed(90, "X")) {Debug.trace("Гав!\n")} //Гав! если нажаты Z, X отдельно или вместе
} //конец цикла
```

GetKeyState

Проверить включена или нет требуемая **Lock**-кнопка клавиатуры.

Синтаксис:

GetKeyState(Key)

Возврат:

(Б), состояние **Lock**-кнопки (**true** - включена, **false** - выключена).

Параметры:

Key - **(С)**, обозначение нужной **Lock**-кнопки: **"N"** - **Num Lock**, **"C"** - **Caps Lock**, **"S"** - **Scroll Lock**.

Пример:

```
if (GetKeyState("C")) {Debug.trace("Caps Lock сейчас включен\n")}
else {Debug.trace("Caps Lock сейчас выключен\n")}
```

SetKeyState

Включить/выключить нужную **Lock**-кнопку клавиатуры.

Синтаксис:

SetKeyState(Key, State)

Возврат:

(Б), изменение состояния работы **Lock**-кнопки (**true** - изменилось, **false** - уже находится в указанном состоянии).

Параметры:

Key - **(С)**, обозначение **Lock**-кнопки: **"N"** - **Num Lock**, **"C"** - **Caps Lock**, **"S"** - **Scroll Lock**.
State - **(Б)**, состояние **Lock**-кнопки, **true** - включить, **false** - выключить.

Пример:

```
if (SetKeyState("C", true)) {Debug.trace("Caps Lock был включен\n")}
else {Debug.trace("Caps Lock уже включенный\n")}
```

GetMousePosition

Получить текущие координаты курсора мыши.

Синтаксис:

```
GetMousePosition()
```

Возврат:

(О) со следующими свойствами:

x - **(Ц)**, текущая **X**-координата курсора мыши относительно верхнего левого угла страницы.

y - **(Ц)**, текущая **Y**-координата курсора мыши относительно верхнего левого угла страницы.

Пример:

```
while (true) {wait(1) //вывод координат курсора мыши будет обновляться каждую секунду
MousePos=GetMousePosition() //создание переменной для хранения объекта с координатами мыши
Debug.trace("X-координата: "+MousePos.x+"\n") //вывод X-координаты мыши в консоль
Debug.trace("Y-координата: "+MousePos.y+"\n")} //вывод Y-координаты мыши в консоль
```

IsMousePressed

Получить текущее состояние нажатия кнопок мыши. **ВАЖНО**: функция **НЕ проверяет комбинацию** из нажатых кнопок, а проверяет нажата ли хоть одна кнопка из указанных.

Синтаксис:

```
IsMousePressed(Button1, Button2, ...)
```

Возврат:

(Б), состояние кнопок мыши (**true** - хотя бы одна из указанных кнопок мыши нажата, иначе **false**).

Параметры:

ButtonN - **(С)**, обозначение кнопки мыши. **ОП**. Возможные значения:
"Left" (левая кнопка мыши), "Right" (правая кнопка мыши), "Middle" (средняя кнопка мыши).

Пример:

```
//Проверка нажатия комбинации кнопок мыши:
while (true) {wait(0.1);LM=IsMousePressed("Left");RM=IsMousePressed("Right")
if (LM==true && RM==false)
{Debug.trace("Левая кнопка мыши сейчас нажата, правая нет\n")}
if (LM==false && RM==true)
{Debug.trace("Правая кнопка мыши сейчас нажата, левая нет\n")}
if (LM==true && RM==true)
{Debug.trace("Левая и правая кнопки мыши сейчас нажаты вместе\n")}}
```

MouseMove

Переместить курсор мыши.

Синтаксис:

```
MouseMove(XPos, YPos)
```

Параметры:

XPos - **(Ц)**, положительное или отрицательное, на сколько изменится текущая **X**-координата мыши. **ОП**.
YPos - **(Ц)**, положительное или отрицательное, на сколько изменится текущая **Y**-координата мыши. **ОП**.

Пример:

```
wait(3);MouseMove(100, -100) //курсор мыши переместится вправо-вверх через 3 секунды
```

MouseClick

Выполнить имитацию нажатия кнопки мыши.

Синтаксис:

```
MouseClick(???)
```

Параметры:

??? - принимаемые параметры неизвестны, есть вероятность, что функция нерабочая.

Базовые объекты

Функции базовых объектов используются со всеми объектами, отображаемыми во вкладке **"Objects"** в органайзере. Названия объектов, созданных в органайзере по умолчанию содержат пробелы (например вновь созданный векторный объект будет называться **"Vector 1"**). При использовании функций базовых объектов важно точное название объекта, такое какое оно есть в органайзере (включая пробелы). При использовании некоторых других функций с объектами недопустимо содержание пробела в названии объекта. Поэтому рекомендуется самостоятельно давать названия всем объектам в органайзере или переименовывать их так, чтобы пробелов в названии не было.
ВАЖНО: на объекты **страницы**, **главы** и **публикации** ссылаться напрямую так, как на остальные объекты, созданные в органайзере **НЕЛЬЗЯ**, например:
Page1.Function() - не сработает, а **GetPage("Page1").Function()** сработает.
Chapter1.Function() - не сработает, а **GetPage("Page1").GetParent().Function()** сработает.
Publication1.Function() - не сработает, а **GetPublication().Function()** сработает.

Список функций:
GetName - получить название объекта.
GetType - получить тип объекта.
GetPage - получить страницу на которой находится объект.
GetUniqueObjectID - получить уникальный идентификатор объекта.
GetResource - получить информацию о ресурсе объекта (возможно нерабочая функция).
FindChild - найти дочерний элемент объекта с определенным названием.
FindDescendant - найти потомка объекта с определенным названием.
GetNumberChildren - получить число дочерних элементов объекта.
GetParent - получить родительский элемент объекта.
GetChild - получить дочерний элемент объекта по индексу.
GetFirstChild - получить первый дочерний элемент объекта.
GetNextChild - получить следующий дочерний элемент объекта.

Для проверки примеров к функциям базовых объектов необходимо создать в органайзере объект кадра с названием **Frame1**. Внутри объекта кадра нужно создать простой векторный объект с названием **Vector1**, содержащий внутри два полигона с названиями **Polygon1** и **Polygon2**.

GetName

Получить название объекта, заданное в органайзере
Синтаксис:
GetName()
Возврат:
(C), название указанного объекта.
Пример:
```
Debug.trace("Название объекта: "+this.GetName())
```

GetType

Получить тип объекта.
Синтаксис:
GetType()
Возврат:
(C), тип указанного объекта. Возможные значения:
Неграфические объекты: **"Publication"**,**"Chapter"**,**"Page"**,**"Timeline"**,**"Script"**,**"ObjectList"**.
Графические объекты: **"Frame"**, **"Button"**, **"Image"**, **"Text"**, **"Video"**, **"Slideshow"**, **"Vector"**, **"Path"**, **"Scrollbar"**, **"Browser"**, **"Frameset"**, **"Rollover"**, **"ListBox"**.
Полигон в векторном объекте: **"Polygon"**.
Неизвестный тип объекта: **"Unknown"**.
Пример:
```
Debug.trace("Тип объекта: "+this.GetType())
```

GetPage

Получить страницу, которой принадлежит указанный объект. При использовании с объектом окна возвращает страницу, отображаемую в этом окне.

Синтаксис:

GetPage()

Возврат:

(O), страница, на которой находится объект.

Пример:

```
Debug.trace("Кадр находится на странице: "+Frame1.GetPage())
```

GetUniqueObjectID

Получить уникальный идентификатор объекта.

Синтаксис:

GetUniqueObjectID()

Возврат:

(C), уникальный идентификатор указанного объекта, имеющий вид **"ObjectXX_XXXX_XXXXXXXX"**, где **X** - это цифры или буквы **A**-**F**.

Пример:

```
Debug.trace("Уникальный идентификатор: "+this.GetUniqueObjectID())
```

GetResource

Получить информацию о ресурсе объекта.

Синтаксис:

GetResource(Resource)

Возврат:

(O) со свойствами ресурса (функция **не работает**, возвращает **null**).

Параметры:

Resource - **(C)**, название ресурса. **ОП**.

Пример:

```
Debug.trace(Frame1.GetResource(SYSTEM_PUBLICATION_DIR+"Resource\\Picture.png")
```

FindChild

Получить конкретный дочерний элемент объекта.

Синтаксис:

FindChild(Name)

Возврат:

(O), дочерний элемент указанного объекта если он существует, иначе **null**.

Параметры:

Name - **(C)**, название дочернего элемента. **ОП**.

Пример:

```
Debug.trace(Vector1.FindChild("Polygon1")) //результат: [ILMObject Polygon1]
```

FindDescendant

Получить конкретного потомка объекта.

Синтаксис:

FindDescendant(Name)

Возврат:

(O), потомок (как дочерний элемент так и дочерний элемент дочерних элементов) указанного объекта если он существует, иначе **null**.

Параметры:

Name - **(C)**, название потомка. **ОП**.

Пример:

```
Debug.trace(Frame1.FindDescendant("Polygon2")) //результат: [ILMObject Polygon2]
```

GetNumberChildren

Получить количество дочерних элементов объекта.

Синтаксис:

GetNumberChildren()

Возврат:

(Ц), количество дочерних элементов указанного объекта (дочерние элементы дочерних не считаются). В число дочерних элементов не входят скриптовые объекты.

Пример:

```
Debug.trace(Vector1.GetNumberChildren()) //результат: 2
```

GetParent

Получить родительский элемент объекта.

Синтаксис:

GetParent()

Возврат:

(О), родительский элемент указанного объекта (содержащий внутри себя проверяемый объект). Публикация (высший в иерархии объект) не имеет над собой родительского элемента, поэтому если использовать на ней эту функцию, то **null**.

Пример:

```
Debug.trace(Vector1.GetParent()) //результат: [ILMObject Frame1]
```

GetChild

Получить по номеру дочерний элемент объекта.

Синтаксис:

GetChild(Index)

Возврат:

(О), конкретный дочерний элемент указанного объекта. Если не существует дочернего элемента под указанным номером, то **null**.

Параметры:

Index - **(Ц)**, порядковый номер дочернего элемента (нумерация начинается с **0**). **ОП**. Самый нижний дочерний элемент объекта в органайзере является первым (под номером **0**).

Пример:

```
Debug.trace(Vector1.GetChild(1)) //результат: [ILMObject Polygon2]
```

GetFirstChild

Получить первый дочерний элемент объекта.

Синтаксис:

GetFirstChild()

Возврат:

(О), первый дочерний элемент указанного объекта. Если у объекта нет дочерних элементов, то **null**.

Пример:

```
Debug.trace(Vector1.GetFirstChild()) //результат: [ILMObject Polygon1]
```

GetNextChild

Получить следующий дочерний элемент объекта.

Синтаксис:

GetNextChild(Before)

Возврат:

(О), дочерний элемент заданного объекта, который следует после указанного в функции другого дочернего. Если не существует такого дочернего элемента, либо он последний, то **null**.

Параметры:

Before - **(О)**, дочерний элемент объекта, находящийся перед требуемым. **ОП**.

Пример:

```
Debug.trace(Vector1.GetNextChild(Polygon1)) //результат: [ILMObject Polygon2]
```

Публикации

Публикация - высший и главный в иерархии объект (глобальный).

Список функций:

ChangeDisplayMode - изменить разрешение экрана во время работы публикации.
GetPublication - получить объект текущей публикации.
ResetVariables - сбросить все переменные публикации до значений по умолчанию.
SavePublicationState - сохранить состояние публикации в файл.
RestorePublicationState - загрузить состояние публикации из файла.
IsPreview - проверить является ли запуск публикации из редактора программы.
ReallyExitPublication - выйти из публикации незамедлительно.
ExitPublication - перейти на страницу выхода из публикации.
TimeGetSeconds - получить количество секунд работы запущенной публикации.
GetPublicationKey - получить оценочный регистрационный ключ для публикации.
TestPublicationKey - протестировать строку с регистрационным ключом пробного периода публикации.
UpdateEvalState - обновить текущий статус пробного периода публикации.
ReadRegistryKey - получить значение указанного ключа в системном реестре.
WriteRegistryValue - записать пару имя/значение в системный реестр.
DeleteRegistryValue - удалить пару имя/значение из системного реестра.
GetLocale - получить текущий язык операционной системы.

Главы

Функции главы предназначены для управления окнами в публикации.

Список функций:

SetInitialPositionExact - установить координаты предварительного расположение окна на экране.
SetInitialPositionPercent - установить процентное предварительное расположение окна на экране.
GetWindowState - получить свойства (координаты и размер) окна.
GetType - получить тип окна.
GetView - получить текущее окно как объект.
GetViewAt - получить окно по номеру как объект.
GetFirstView - получить первое открытое окно как объект.
GetNextView - получить следующее открытое окно как объект.
GetNumberViews - получить количество открытых окон.
Close - закрыть окно.

ChangeDisplayMode

Изменить разрешение экрана и глубину цвета, пока работает запущенная публикация.

Синтаксис:

ChangeDisplayMode(DisplayIndex, DepthIndex)

Возврат:

(Б), состояние экрана (**true** - разрешение изменилось, **false** - не изменилось).

Параметры:

DisplayIndex - **(Ц)**, режим дисплея для изменения разрешения экрана. **НП**, по умолчанию **-1** (разрешение экрана устройства, как оно было до запуска публикации). Возможные режимы: **0** - **640x480**, **1** - **800x600**, **2** - **1024x768** и т.д.

DepthIndex - **(Ц)**, глубина цвета на экране. **НП**, по умолчанию **-1** (глубина цвета на экране устройства, как она была до запуска публикации).

Возможные значения: **15**, **16**, **24**, **36**.

Пример:

```
ChangeDisplayMode(1) //изменить разрешение экрана на 800x600
```

GetPublication

Получить текущую публикацию как объект.

Синтаксис:

GetPublication()

Возврат:

(О), текущая публикация.

Пример:

```
Debug.trace("Публикация: "+GetPublication())
```

ResetVariables

Сбросить значения всех переменных публикации до значений по умолчанию. Переменные страницы не сбрасываются, для сброса переменных страницы существует специальная функция **ResetVars**.

Объявление переменных публикации: пункт меню **"Edit"** > команда **"Publication Properties"** > вкладка **"Variables"**.

Синтаксис:

ResetVariables()

Пример:

```
ResetVariables()
```

SavePublicationState и RestorePublicationState

Сохранить/загрузить текущее состояние публикации.

Синтаксис:

SavePublicationState(Filename)

RestorePublicationState(Filename)

Возврат:

(Б), результат сохранения/загрузки (**true** - процесс прошел успешно, иначе **false**).

Параметры:

Filename - **(С)**, путь к файлу для сохранения/загрузки. **ОП**.

Пример:

```
SavePublicationState(SYSTEM_PUBLICATION_DIR+"State0.sav")
wait(3); RestorePublicationState(SYSTEM_PUBLICATION_DIR+"State0.sav")
```

IsPreview

Проверить, запущена ли публикация в редакторе программы или нет.

Синтаксис:

IsPreview()

Возврат:

(Б), результат проверки (**true** - запущена публикация из редактора программы, **false** - запущена скомпилированная публикация).

Пример:

```
Debug.trace("Публикация запущена из редактора программы: "+IsPreview())
```

ExitPublication и ReallyExitPublication

Выйти из публикации (**ReallyExitPublication** - сразу, игнорируя страницу выхода, **ExitPublication** - с переходом на страницу выхода).

1) Назначение страницы выхода: пункт меню **"Publication"** > команда **"Publication Properties"** > вкладка **"Options"** > флажок **"On Exit Display Page"**.

2) Если страница выхода не назначена, либо функция выхода вызывается на самой странице выхода, то выход из публикации происходит сразу.

Синтаксис:

ReallyExitPublication()

ExitPublication()

Пример:

```
ReallyExitPublication()
ExitPublication()
```

TimeGetSeconds

Получить количество секунд работы текущей публикации с момента её запуска.

Синтаксис:

TimeGetSeconds()

Возврат:

(Ч), количество секунд работы текущей публикации с момента её запуска.

Пример:

```
wait(3.6);Debug.trace("Публикация работает "+TimeGetSeconds()+" секунд")
```

GetPublicationKey

Получить ключ пробной публикации.

Необходимо включить опцию **"Registration ID"** на вкладке **"Security"** параметров публикации.

Синтаксис:

GetPublicationKey()

Возврат:

(С), восьмизначный ключ оценки публикации, состоящий из цифр и латинских букв **a-f**.

Пример:

```
Debug.trace("Ключ публикации: "+GetPublicationKey())
```

TestPublicationKey

Проверить действительность ключа пробной публикации.

Необходимо включить опцию **"Registration ID"** на вкладке **"Security"** параметров публикации.

Синтаксис:

TestPublicationKey(Test)

Возврат:

(Б), действительность ключа разблокировки (**true** - ключ действителен, **false** - нет).

Параметры:

Test - **(С)**, код разблокировки для тестирования. **ОП**.

Пример:

```
Debug.trace(TestPublicationKey("3190c6ee"))
```

UpdateEvalState

Обновить текущее состояние пробного периода публикации. Полезно при работе по истечении срока пробного периода. Обновление состояния пробного периода не приведет к автоматическому истечению срока публикации или переходу на какую-либо страницу с истекшим сроком действия. Просто обновляется системная переменная **PUBLICATION_EVALUATION** с правильной информацией.

Синтаксис:

UpdateEvalState()

Пример:

```
UpdateEvalState()
```

ReadRegistryKey

Получить значение из реестра.

Синтаксис:

ReadRegistryKey(RegistryPath, Key)

Возврат:

(С), значение параметра из ветви реестра, если параметра не существует, то пустое строковое значение.

Параметры:

RegistryPath - **(С)**, полный путь ветви рееста. **ОП**.

Key - **(С)**, название параметра из ветви реестра. **ОП**.

Пример:

```
var RegPath="HKEY_LOCAL_MACHINE\\SOFTWARE\\Digital Workshop\\Opus Professional 9"
Debug.trace("Версия программы: "+ReadRegistryKey(RegPath, "Version"))
```

WriteRegistryValue

Записать значение в реестр.

Только в **HKEY_LOCAL_MACHINE** или **HKEY_CURRENT_USER** в разделе **Software**.

Синтаксис:

WriteRegistryValue(RegistryPath, Key, Value)

Возврат:

(Б), результат записи (**true** - значение было записано, иначе **false**).

Параметры:

RegistryPath - **(С)**, полный путь ветви реестра. **ОП**.

Key - **(С)**, название параметра из ветви реестра. **ОП**.

Value - **(С)** или **(Ц)**, значение параметра из ветви реестра. **ОП**.

Пример:

```
var RegPath="HKEY_CURRENT_USER\\Software\\Digital Workshop\\Opus Professional 9\\Registration"
var RegName="Rumburak"
var Serial=String.mid(String.toupper(GenerateGUID()), 10, 19)
WriteRegistryValue(RegPath, "RegisteredTo", RegName)
WriteRegistryValue(RegPath, "Code", Serial)
```

DeleteRegistryValue

Удалить значение из реестра.

Только в **HKEY_LOCAL_MACHINE** или **HKEY_CURRENT_USER** в разделе **Software**.

Синтаксис:

DeleteRegistryValue(RegistryPath, Key)

Возврат:

(Б), результат удаления (**true** - значение было удалено, иначе **false**).

Параметры:

RegistryPath - **(С)**, полный путь ветви реестра. **ОП**.

Key - **(С)**, название параметра из ветви реестра. **ОП**.

Пример:

```
var RegPath="HKEY_CURRENT_USER\\Software\\Digital Workshop\\Opus Professional 9\\Registration"
DeleteRegistryValue(RegPath, "RegisteredTo")
```

GetLocale

Получить язык операционной системы (**НЕ** язык ввода с клавиатуры).

Синтаксис:

GetLocale(Info)

Параметры:

Info - **(Ц)**, тип информации. **НП**, по умолчанию **0**. Возможные значения:

0 - трехбуквенный код страны, **1** - двухбуквенный код страны, **2** - двухбуквенный код языка.

Возврат:

(С), язык операционной системы.

Пример:

```
Debug.trace("Язык системы: "+GetLocale(1))
```

SetInitialPositionExact и SetInitialPositionPercent

Установить предварительное расположение окна на экране, но не изменять расположение уже открытого окна.

Синтаксис:

SetInitialPositionExact(PosX, PosY)
SetInitialPositionPercent(PercentX, PercentY)

Параметры:

PosX - **(Ц)**, **X**-координата левого края рамки окна, относительно экрана. **-1** для исходного состояния. **ОП**.
PosY - **(Ц)**, **Y**-координата верхнего края рамки окна, относительно экрана. **-1** для исходного состояния. **ОП**.
PercentX - **(Ц)**, процентное расположение окна на экране по горизонтали. **-1** для исходного состояния. **ОП**.
PercentY - **(Ц)**, процентное расположение окна на экране по вертикали. **-1** для исходного состояния. **ОП**.

Пример:

```
var Chap=GetPublication().FindChild("Chapter 2") //найти объект главы для изменения
Chap.SetInitialPositionExact(200, 100) //установить положение окна
GotoPage(Chap.GetFirstChild().GetName()) //открыть главу, перейдя на первую страницу в ней
```

GetWindowState

Получить свойства окна.

Синтаксис:

GetWindowState()

Возврат:

(О) со следующими свойствами указанного окна:
x - **(Ц)**, **X**-координата левого края содержимого в окне, относительно экрана.
y - **(Ц)**, **Y**-координата верхнего края содержимого в окне, относительно экрана.
width - **(Ц)**, ширина содержимого в окне.
height - **(Ц)**, высота содержимого в окне.
wnd_x - **(Ц)**, **X**-координата левого края рамки окна, относительно экрана.
wnd_y - **(Ц)**, **Y**-координата верхнего края рамки окна, относительно экрана.
wnd_width - **(Ц)**, ширина рамки окна.
wnd_height - **(Ц)**, высота рамки окна, включая заголовок.
bMaximized - **(Б)**, **true** если окно в данный момент развернуто, иначе **false**.
bMinimized - **(Б)**, **true** если окно в данный момент свернуто, иначе **false**.

Пример:

```
var CurrentView=GetView(); var WinProp=CurrentView.GetWindowState()
Debug.trace("X-координата содержимого в окне: "+WinProp.x+"\n")
Debug.trace("Y-координата содержимого в окне: "+WinProp.y+"\n")
Debug.trace("Ширина содержимого в окне: "+WinProp.width+"\n")
Debug.trace("Высота содержимого в окне: "+WinProp.height+"\n")
Debug.trace("X-координата рамки окна: "+WinProp.wnd_x+"\n")
Debug.trace("Y-координата рамки окна: "+WinProp.wnd_y+"\n")
Debug.trace("Ширина рамки окна: "+WinProp.wnd_width+"\n")
Debug.trace("Высота рамки окна: "+WinProp.wnd_height+"\n")
Debug.trace("Окно максимизировано? "+WinProp.bMaximized+"\n")
Debug.trace("Окно минимизировано? "+WinProp.bMinimized+"\n")
```

GetType

Получить тип окна.

Синтаксис:

GetType()

Возврат:

(Ц), тип указанного окна, возможные значения:
0 - главное окно; **1**, **2**, **3**, **4** - панель с номером; **-1** - внешнее окно; **-2** - ошибка.

Пример:

```
var View1=GetView(); Debug.trace("Тип окна: "+View1.GetType())
```

GetView

Получить текущее окно.

Синтаксис:

GetView()

Возврат:

(О), текущее окно.

Пример:

```
var View1=GetView()
```

GetViewAt

Получить окно под указанным номером.

Синтаксис:

GetViewAt(Index)

Возврат:

(О), окно под указанным номером (**null** если такого окна не существует).

Параметры:

Index - **(Ц)**, порядковый номер окна (нумерация начинается с **0**). **ОП**.

Пример:

```
var View1=GetViewAt(0)
```

GetFirstView

Получить первое из открытых на данный момент окон.

Синтаксис:

GetFirstView()

Возврат:

(О), первое окно из открытых на данный момент.

Пример:

```
var View1=GetFirstView()
```

GetNextView

Получить следующее за указанным окно из открытых на данный момент.

Синтаксис:

GetNextView(View)

Возврат:

(О), открытое окно, следующее за указанным (**null** если такого окна не существует).

Параметры:

View - **(О)**, окно, для получения следующего после него. **ОП**.

Пример:

```
var TestView=GetFirstView(); Debug.trace(GetNextView(TestView))
```

GetNumberViews

Получить количество открытых на данный момент окон.

Синтаксис:

GetNumberViews()

Возврат:

(Ц), количество окон, открытых на данный момент.

Пример:

```
Debug.trace("Количество открытых окон: "+GetNumberViews())
```

Close

Закрыть указанное окно, кроме панелей. Закрытие главного окна приводит к выходу из публикации.

Синтаксис:

Close()

Пример:

```
var View1=GetFirstView(); View1.Close()
```

Страницы

Список функций:

ResetVars - сбросить все переменные текущей страницы до значений по умолчанию.
GotoRandomPage - перейти на случайную страницу главы, включая уже посещенные.
GotoNextRandomPage - перейти на случайную страницу главы, исключая уже посещенные.
GotoPage - перейти на указанную страницу любой главы.
GotoForwardPage - перейти на следующую страницу текущей главы.
GotoBackwardPage - перейти на предыдущую страницу текущей главы.
GotoNextPage - перейти на следующую страницу из истории посещенных страниц.
GotoPreviousPage - перейти на предыдущую страницу из истории посещенных страниц.
GotoCurrentPage - перейти на текущую страницу из истории посещенных страниц.
GetPageNumber - получить номер текущей страницы.
GetPage - получить указанную страницу как объект.
GetLastPage - получить последнюю страницу как объект.
GetNextPage - получить следующую страницу как объект.
GetPreviousPage - получить предыдущую страницу как объект.
GetPageDownloadTotal - получить общий объем загруженной страницы.
GetPageDownloadPosition - получить позицию загруженной страницы.
GetPageDownloadPercent - получить процент загрузки страницы.
CopyToClipboard - скопировать страницу в буфер обмена **Windows**.
PrintPage - распечатать страницу.
ShowBookmarkDialog - показать окно закладок **"Bookmarks"**.
SetBookmark - назначить закладку для текущей страницы.
ClearBookmark - убрать закладку с текущей страницы.
GotoBookmark - перейти к ближайшей странице с закладкой.
GetBookmarkPage - получить страницу с закладкой по номеру.
GetFirstBookmark - получить первую страницу с закладкой.
GetNextBookmark - получить следующую страницу с закладкой.

ResetVars

Сбросить значения всех переменных текущей страницы до значений по умолчанию. Переменные публикации не сбрасываются, для сброса переменных публикации существует специальная функция **ResetVariables**. Переменные объявленные в скриптовом объекте **НЕ сбрасываются**. Объявление переменных страницы: пункт меню **"Edit"** > команда **"Page Properties"** > вкладка **"Variables"**.

Синтаксис:

ResetVars()

Пример:

```
ResetVars()
```

GotoRandomPage

Перейти на случайную страницу текущей главы (включая уже посещенные), отличную от текущей страницы. Если страница одна, то происходит её сброс в исходное состояние.

Синтаксис:

GotoRandomPage()

Пример:

```
GotoRandomPage()
```

GotoNextRandomPage

Перейти на случайную страницу текущей главы, не переходя на уже посещенные страницы. Когда функция вызывается первый раз, то формируется список всех страниц главы. При последующих вызовах функции переход на посещенные страницы из списка не происходит. Когда все страницы списка посещены, то он сбрасывается в исходное состояние. Если страница одна, то происходит её сброс в исходное состояние.

Синтаксис:

GotoNextRandomPage()

Пример:

```
GotoNextRandomPage()
```

GotoPage

Перейти на указанную страницу любой главы.

Синтаксис:

GotoPage(Page)

Параметры:

Page - **(C)**, название страницы для перехода (порядковый номер страницы не действует). **ОП**.

Пример:

```
GotoPage("Page01")
```

GotoForwardPage

Перейти на следующую страницу текущей главы.

Синтаксис:

GotoForwardPage()

Пример:

```
GotoForwardPage()
```

GotoBackwardPage

Перейти на предыдущую страницу текущей главы.

Синтаксис:

GotoBackwardPage()

Пример:

```
GotoBackwardPage()
```

GotoNextPage

Перейти на следующую страницу любой главы из истории страниц. Во время работы публикации список всех посещенных страниц сохраняется в истории. Если одновременно открыто более одного вида страниц в отдельных окнах, то каждый вид имеет свою собственную историю страниц.

Синтаксис:

GotoNextPage()

Пример:

```
GotoNextPage()
```

GotoPreviousPage

Перейти на предыдущую страницу любой главы из истории страниц. Во время работы публикации список всех посещенных страниц сохраняется в истории. Если одновременно открыто более одного вида страниц в отдельных окнах, то каждый вид имеет свою собственную историю страниц.

Синтаксис:

GotoPreviousPage()

Пример:

```
GotoPreviousPage()
```

GotoCurrentPage

Перейти на текущую страницу публикации из истории страниц. Используется для сброса страницы в исходное состояние. Во время работы публикации список всех посещенных страниц сохраняется в истории. Если одновременно открыто более одного вида страниц в отдельных окнах, то каждый вид имеет свою собственную историю страниц.

Синтаксис:

GotoCurrentPage()

Пример:

```
GotoCurrentPage()
```

GetPageNumber

Получить порядковый номер текущей страницы. Нумерация страниц сквозная (проходит через разные главы) и начинается с **1** (а не с **0**, как у функции **GetPage**). Мастер-страницы не учитываются.

Синтаксис:

GetPageNumber()

Возврат:

(Ц), порядковый номер страницы, на которой вызвана функция.

Пример:

```
Debug.trace("Номер страницы: "+GetPageNumber())
```

GetPage

Получить конкретную страницу для использования с другими функциями.

Синтаксис:

GetPage(Name)

Возврат:

(О), указанная страница.

Параметры:

Name - **(С)**, название страницы, либо **(Ц)**, порядковый номер страницы. Нумерация страниц сквозная (проходит через разные главы) и начинается с **0** (а не с **1**, как у функции **GetPageNumber**). **ОП**.

Пример:

```
//В свежесозданной публикации страница с названием "Page 1" имеет номер 0
Debug.trace(GetPage("Page 1")+"\n"+GetPage(0))
```

GetLastPage

Получить последнюю страницу для использования с другими функциями.

Синтаксис:

GetLastPage()

Возврат:

(О), последняя страница (всей публикации).

Пример:

```
Debug.trace("Последняя страница: "+GetLastPage())
```

GetNextPage

Получить следующую страницу для использования с другими функциями.

Синтаксис:

GetNextPage()

Возврат:

(О), следующая страница, если следующей страницы не существует (например после последней), то **null**.

Пример:

```
Debug.trace(GetNextPage())
```

GetPreviousPage

Получить предыдущую страницу для использования с другими функциями.

Синтаксис:

GetPreviousPage()

Возврат:

(О), предыдущая страница, если предыдущей страницы не существует (например перед первой), то **null**.

Пример:

```
Debug.trace(GetPreviousPage())
```

GetPageDownloadTotal

Получить общий объем загруженной страницы.

Синтаксис:

GetPageDownloadTotal(Page)

Возврат:

(Ц), общий объем загруженной страницы.

Параметры:

Page - **(С)**, название страницы, проверяемой на информацию о загрузке. **ОП**.

Пример:

```
Debug.trace(GetPageDownloadTotal("Page01"))
```

GetPageDownloadPosition

Получить позицию загруженной страницы.

Синтаксис:

GetPageDownloadPosition(Page)

Возврат:

(Ц), позиция указанной загруженной страницы.

Параметры:

Page - **(С)**, название страницы, проверяемой на информацию о загрузке. **ОП**.

Пример:

```
Debug.trace(GetPageDownloadPosition("Page01"))
```

GetPageDownloadPercent

Получить процент загрузки страницы.

Синтаксис:

GetPageDownloadPercent(Page)

Возврат:

(Ц), процент загрузки указанной страницы.

Параметры:

Page - **(С)**, название страницы, проверяемой на информацию о загрузке. **ОП**.

Пример:

```
Debug.trace(GetPageDownloadPercent("Page01"))
```

CopyToClipboard

Скопировать изображение текущей страницы в буфер обмена **Windows**.

Синтаксис:

CopyToClipboard()

Пример:

```
CopyToClipboard()
```

PrintPage

Распечатать страницу.

Синтаксис:

PrintPage(PageName, PrintDialog, CancelDialog)

Параметры:

PageName - **(С)**, название страницы для печати. **НП**, если страница не указана, то печатается текущая страница.

PrintDialog - **(Б)**, отображение окна свойств печати перед распечаткой (**true** - окно появится, **false** - нет). **НП**, по умолчанию **false**.

CancelDialog - **(Б)**, отображение окна отмены печати во время распечатки (**true** - окно появится, **false** - нет). НП, по умолчанию **true**.

Пример:

```
PrintPage("Page01", true, false)
```

ShowBookmarkDialog

Вывести на экран встроенное окно закладок **"Bookmarks"**.

Синтаксис:

ShowBookmarkDialog()

Пример:

ShowBookmarkDialog()

SetBookmark

Назначить закладку для текущей страницы.

Синтаксис:

SetBookmark()

Пример:

SetBookmark()

ClearBookmark

Убрать закладку с текущей страницы.

Синтаксис:

ClearBookmark()

Пример:

ClearBookmark()

GotoBookmark

Перейти к ближайшей странице с закладкой.

Синтаксис:

GotoBookmark()

Пример:

GotoBookmark()

GetBookmarkPage

Получить страницу с закладкой по номеру.

Синтаксис:

GetBookmarkPage(Index)

Возврат:

(О), страница с закладкой, если такой не существует, то **null**.

Параметры:

Index - **(Ц)**, порядковый номер страницы с закладкой (нумерация начинается с **1**). **ОП**.

Пример:

GetBookmarkPage(1)

GetFirstBookmark

Получить первую страницу с закладкой.

Синтаксис:

GetFirstBookmark()

Возврат:

(О), первая страница с закладкой, если такой не существует, то **null**.

Пример:

GetFirstBookmark()

GetNextBookmark

Получить следующую страницу с закладкой.

Синтаксис:

GetNextBookmark()

Возврат:

(О), следующая страница с закладкой, если такой не существует, то **null**.

Пример:

GetNextBookmark()

Поиск

Функции поиска предназначены для выбора страницы по ключевому слову (ключевые слова с русскими символами не определяются).

Особенности использования:

1) Необходимо в окне свойств публикации **"Publication Properties"** на вкладке **"General(2)"** в поле **"Miscellaneous options"** установить флажок **"Use publication search database"**.

2) Функция **LaunchSearch** предназначена для запуска встроенного окна поиска и не нуждается в дополнительных функциях.

3) Функция **OpenSearch** и остальные предназначены для создания объекта поиска и управления им.

Список функций:

LaunchSearch - отобразить встроенное окно поиска.
OpenSearch - создать объект поиска со списком ключевых слов.
GetFirstKeyword - получить первое ключевое слово из объекта поиска.
GetNextKeyword - получить следующее ключевое слово из объекта поиска.
GetFirstPage - получить первую страницу с заданным ключевым словом.
GetNextPage - получить следующую страницу с заданным ключевым словом.

LaunchSearch

Отобразить на экране встроенное окно поиска в публикации.

Синтаксис:

```
LaunchSearch(PosX, PosY)
```

Параметры:

PosX - **(Ц)**, **X**-координата верхнего левого угла окна поиска. **НП**, по умолчанию **0**.
PosY - **(Ц)**, **Y**-координата верхнего левого угла окна поиска. **НП**, по умолчанию **0**.

Пример:

```
LaunchSearch(100, 100)
```

OpenSearch

Создать объект поиска в публикации.

Синтаксис:

```
OpenSearch()
```

Возврат:

(О) для поиска в публикации со списком ключевых слов.

Пример:

```
var Search1=OpenSearch()
```

GetFirstKeyword

Получить первое ключевое слово в объекте поиска.

Синтаксис:

```
GetFirstKeyword()
```

Возврат:

(С), первое ключевое слово в объекте поиска (**null** если такого ключевого слова нет).

Пример:

```
var Search1=OpenSearch()
Debug.trace("Первое ключевое слово: "+Search1.GetFirstKeyword())
```

GetNextKeyword

Получить следующее ключевое слово в объекте поиска.

Синтаксис:

```
GetNextKeyword()
```

Возврат:

(С), следующее ключевое слово в объекте поиска (**null** если такого ключевого слова нет).

Пример:

```
var Search1=OpenSearch();Search1.GetFirstKeyword()
Debug.trace("Следующее ключевое слово: "+Search1.GetNextKeyword())
```

GetFirstPage

Получить первую страницу, содержащую указанное ключевое слово объекта поиска.

Синтаксис:

GetFirstPage(Keyword)

Возврат:

(O), первая страница, содержащая указанное ключевое слово (**null** если такой страницы нет).

Параметры:

(C), ключевое слово. **ОП**.

Пример:

```
var Search1=OpenSearch()
var Keyword1=Search1.GetFirstKeyword()
var Page1=Search1.GetFirstPage(Keyword1)
GotoPage(Page1)
```

GetNextPage

Получить следующую страницу, содержащую указанное ключевое слово объекта поиска.

Синтаксис:

GetNextPage(Keyword)

Возврат:

(O), следующая страница, содержащая указанное ключевое слово (**null** если такой страницы нет).

Параметры:

(C), ключевое слово. **ОП**.

Пример:

```
var Search1=OpenSearch()
var Keyword1=Search1.GetFirstKeyword()
Search1.GetFirstPage(Keyword1)
var Page1=Search1.GetNextPage(Keyword1)
GotoPage(Page1)
```

Графические объекты

Основные графические объекты:
Frame (кадр) - контейнер, в котором могут располагаться любые другие гр. объекты, включая сами кадры.
Multiframe (мультикадр) - переключатель между несколькими кадрами.
Text (текстовый объект) - гр. объект, содержащий текст.
Image (изображение) - гр. объект, состоящий лишь из фонового растрового изображения.
Vector - (векторный объект) гр. объект, содержащий векторную графику.
Video (видео) - гр. объект, содержащий видеофайл.

Специальные графические объекты:
Tween (анимация) - специальный кадр для анимации.
Question (вопрос) - специальный кадр для вопросов.
Rollover - специальный мультикадр с отдельным кадром на каждое состояние.
Text Input Box (поле ввода текста) - текстовый объект, специализированный для ввода текста.
Listbox (список) - текстовый объект, специализированный для создания списка.
Button (кнопка) - изображение с внутренним текстовым объектом, специализированное для использования как кнопка.
Slideshow (слайд-шоу) - набор переключаемых изображений.
Hotspot (активная зона) - специальный векторный объект, всегда невидимый при запуске публикации.
Vertical/Horizontal Scrollbar - полоса покрутки с ползунком, состоит из изображения и двух кнопок.

Эффект **Blend** для создания масок не задается скриптом. Маски влияют на все остальные эффекты. Для создания эффекта необходимо: в свойствах гр. объекта **"Properties"** на вкладке **"Effects"** в поле **"Category"** поставить флажок **"Blend"** и настроить. Черный цвет применяемого изображения - невидимая область, белый - видимая, остальные цвета - различная степень прозрачности.
Цвет в параметрах некоторых функций задаётся одним из следующих способов:
(С), один из восьми цветов (**"white"**, **"black"**, **"red"**, **"green"**, **"blue"** , **"yellow"**, **"cyan"**, **"magenta"**);
(С), шестнадцатеричный код цвета **HTML**, вида **"#000000"**, где знак **#** обязателен, а код состоит из цифр и букв **A**-**F**;
(Ц), полученное при помощи функции **RGB**.

Список функций:
Enable - включить или отключить гр. объект.
IsEnabled - проверить, включен ли гр. объект.
Show - показать гр. объект.
Hide - скрыть гр. объект.
IsShowing - проверить, показывается ли гр. объект.
GetLayer - получить слой, на котором находится гр. объект.
SetLayer - установить слой, на котором находится гр. объект.
CloneObject - клонировать гр. объект.
DestroyClonedObject - удалить клонированный гр. объект.
GetPersistentObject - получить объект для хранения информации о гр. объекте.
IsObjectIntersecting - проверить, пересекается ли гр. объект с другим.
GetWidth - получить ширину гр. объекта.
GetHeight - получить высоту гр. объекта.
SetObjectSize - установить ширину и высоту гр. объекта.
GetObjectDimensions - получить положение и размер гр. объекта.
SetObjectDimensions - установить положение и размер гр. объекта.
GetDisplayData - получить свойства трансформации гр. объекта.
SetDisplayData - установить свойства трансформации гр. объекта.
GetPosition (для всех гр. объектов) - получить координаты положения гр. объекта.
GetXPosition - получить **X**-координату положения гр. объекта.
GetYPosition - получить **Y**-координату положения гр. объекта.
SetPosition - установить координаты положения гр. объекта.
SetPositionX - установить **X**-координату положения гр. объекта.
SetPositionY - установить **Y**-координату положения гр. объекта.
Move - переместить гр. объект.

MoveX - переместить гр. объект по **X**-оси.
MoveY - переместить гр. объект по **Y**-оси.
GetPosition (для пути) - получить координаты точки на анимационном пути по пиксельному расстоянию.
GetPositionFromPercent - получить координаты точки на анимационном пути по процентному расстоянию.
GetTotalLength - получить длину векторной линии объекта анимационного пути.
FollowPath - выполнить движение гр. объекта по анимационному пути.
SetTransparency - установить прозрачность гр. объекта.
Fade - изменить прозрачность гр. объекта.
SetRotation - установить поворот гр. объекта вокруг центра.
Rotate - повернуть гр. объект вокруг центра.
SetRoll - назначить поворот гр. объекта вокруг **X**-оси.
Roll - повернуть гр. объект вокруг **X**-оси.
SetSpin - назначить поворот гр. объекта вокруг **Y**-оси.
Spin - повернуть гр. объект вокруг **Y**-оси.
SetScale - установить горизонтальный и вертикальный масштаб гр. объекта в процентах.
SetScaleH - установить горизонтальный масштаб гр. объекта в процентах.
SetScaleV - установить вертикальный масштаб гр. объекта в процентах.
Scale - масштабировать гр. объект по горизонтали и вертикали.
ScaleH - масштабировать гр. объект по горизонтали.
ScaleV - масштабировать гр. объект по вертикали.
SetSkew - наклонить гр. объект горизонтально и вертикально.
SetSkewH - наклонить гр. объект горизонтально.
SetSkewV - наклонить гр. объект вертикально.
Skew - назначить перекос гр. объекта по горизонтали и вертикали.
SkewH - назначить перекос гр. объекта горизонтально.
SkewV - назначить перекос гр. объекта вертикально.
ResetAnimation - сбросить анимацию гр. объекта.
StopAnimation - остановить анимацию гр. объекта.
GetScrollInfo - получить значения состояния прокрутки.
SetScrollPosition - установить значения состояния прокрутки.
GetAppearance - получить объект внешнего вида для гр. объекта.
ReloadImage - перезагрузить фоновое растровое изображение для гр. объекта.
SetBackground - создать фон для гр. объекта.
RemoveBackground - убрать фон для гр. объекта.
SetImage - создать фоновое растровое изображение для гр. объекта.
RemoveImage - убрать фоновое растровое изображение для гр. объекта.
SetTexture - создать эффект текстуры для гр. объекта.
RemoveTexture - убрать эффект текстуры для гр. объекта.
SetAlpha - создать альфа-эффект для гр. объекта.
RemoveAlpha - убрать альфа-эффект для гр. объекта.
SetShadow - создать эффект тени для гр. объекта.
RemoveShadow - убрать эффект тени для гр. объекта.
SetFlare - создать эффект свечения для гр. объекта.
RemoveFlare - убрать эффект свечения для гр. объекта.
SetBorder - создать рамку для гр. объекта.
RemoveBorder - убрать рамку для гр. объекта.
SetBtnColour - создать эффект кнопки для гр. объекта.
RemoveBtnColour - убрать эффект кнопки для гр. объекта.
SetFocus - направить весь ввод с клавиатуры на гр. объект.
ClearInputFocus - снять с гр. объекта ранее направленный ввод с клавиатуры.
CaptureMouse - направить весь ввод мышью на гр. объект.
ReleaseMouse - снять с гр. объекта ранее направленный ввод мышью.
CopyToClipboard - скопировать гр. объект в буфер обмена.
PrintObject - распечатать гр. объект.

Enable

Включить или отключить графический объект.

Особенности функции:

1) Состояние (**Appearance**) гр. объекта при отключении становится **"Disabled"**, а при включении **"Normal"** .

2) Отключенный гр. объект продолжает отображаться на странице и на него действуют все скриптовые функции, причем отключение гр. объекта **повышает общую производительность** публикации.

3) Действия (**Actions**), назначенные отключенному гр. объекту, перестанут срабатывать на триггеры (**Triggers**), но уже запущенное действие не остановится при отключении гр. объекта. Действия, назначенные другому гр. объекту, но влияющие на отключенный гр. объект, будут срабатывать.

4) Такие гр. объекты как: **кнопки**, **поля ввода текста** и **списки** - станут неактивными.

Синтаксис:

Enable(Set)

Параметры:

Set - **(Б)**, состояние работы гр. объекта (**true** - включить, **false** - отключить). **НП**, по умолчанию **true**.

Примеры:

```
Button1.Enable(false) //кнопка перестанет работать (нажиматься), но будет отображаться на странице
```

IsEnabled

Проверить текущее состояние работы графического объекта, включен ли он или отключен.

Синтаксис:

IsEnabled()

Возврат:

(Б), состояние работы гр. объекта (**true** - включен, **false** - отключен).

Пример:

```
Debug.trace("Состояние работы кнопки: "+Button1.IsEnabled())
```

Show

Показать скрытый графический объект на странице. Если используется спецэффект (**Transition**), то следующая строчка кода выполнится только после его завершения.

Синтаксис:

Show(ResetPosition)

Параметры:

ResetPosition - **(Б)**, положение гр. объекта при появлении после скрытия (**true** - исходное (заданное в редакторе), **false** - на месте скрытия). **НП**, по умолчанию **true**.

Примеры:

```
Vector1.Move(100, 100, 2) //переместить векторное изображение
wait(1);Vector1.Hide() //скрыть векторное изображения
wait(1);Vector1.Show(false) //отобразить векторное изображение на месте скрытия
```

Hide

Скрыть графический объект на странице. Если используется спецэффект (**Transition**), то следующая строчка кода выполнится только после его завершения.

Синтаксис:

Hide()

Пример:

```
Vector1.Hide() //скрыть векторное изображение
```

IsShowing

Проверить текущее состояние отображения графического объекта на странице.

Синтаксис:

IsShowing()

Возврат:

(Б), состояние отображения гр. объекта (**true** - видимый, **false** - скрытый).

Пример:

```
Debug.trace("Векторное изображение № 1 отображается? "+Vector1.IsShowing())
```

GetLayer

Определить на каком слое располагается графический объект.

Синтаксис:

GetLayer()

Возврат:

(Ц), номер слоя, где расположен гр. объект.

Пример:

```
Debug.trace("Слой №: "+Vector1.GetLayer())
```

SetLayer

Поместить графический объект на определенный слой (объекты на слое с **большим** значением отображаются **впереди**, а с **меньшим** - **позади**). Изначально любой гр. объект располагается на **0** слое.

ВАЖНО: внутри кадра гр. объекты имеют собственную нумерацию слоёв.

Синтаксис:

SetLayer(Number)

Параметры:

Number - **(Ц)** положительное или отрицательное, номер слоя, куда следует поместить гр. объект. **ОП**.

Пример:

```
Vector1.SetLayer(10) //данное векторное изображение будет располагаться впереди
Vector2.SetLayer(-8) //данное векторное изображение будет располагаться позади
```

CloneObject

Создать идентичную копию (клон) графического объекта с прикреплёнными к нему действиями и скриптовым объектом.

Особенности функции:

1) Клон всегда создаётся на слое **0**.

2) Клон будет принадлежать тому же кадру, что и оригинал.

3) Клон имеет тот внешний вид, что был у оригинала в редакторе программы. Если оригинал был подвергнут изменениям с помощью скрипта, а затем клонирован, то эти изменения не передадутся клону.

ИСКЛЮЧЕНИЯ: скриптовые изменения полигонов оригинального векторного объекта повлияют и на клон. Клон кадра **НЕ** будет содержать фигуры оригинала, нарисованные с помощью **Draw**-функций.

4) Чтобы избежать ошибки, не следует клонировать гр. объект из прикрепленного к нему самому скриптового объекта.

5) Чем больше гр. объектов в публикации, тем сильнее снижается её производительность, поэтому чрезмерное число клонов вызовет зависание (общее число гр. объектов в публикации не должно превышать **3168**).

Синтаксис:

CloneObject(PosX, PosY, Visible)

Возврат:

(О), новый клон гр. объекта.

Параметры:

PosX - **(Ц)**, **X**-координата клона. **ОП** (**НП**, если **Y**-координата также не указана, по умолчанию как у оригинала).

PosY - **(Ц)**, **Y**-координата клона. **ОП** (**НП**, если **X**-координата также не указана, по умолчанию как у оригинала).

Если оригинал находится вне кадра, то вводятся координаты относительно верхнего левого угла страницы.

Если оригинал находится внутри кадра, то вводятся координаты относительно верхнего левого угла кадра.

Visible - **(Б)**, видимость клона (**true** - клон видимый, **false** - скрытый). **НП**, по умолчанию как у оригинала.

Пример:

```
var Clone1=Vector1.CloneObject(100, 100)
```

DestroyClonedObject

Удалить клон графического объекта (оригинал удалить нельзя).
ВАЖНО: нельзя удалять клон из основного скрипта пока на него действует прикрепленный скриптовый объект, иначе появится сообщение об ошибке: **"Reference to missing object"**. Рекомендуется удалять клон из прикрепленного к нему скриптового объекта.
Синтаксис:
DestroyClonedObject()
Возврат:
(Б), результат удаления (**true** - клон удален, иначе **false**).
Пример:

```
//***Удаление клона без вывода ошибки***
//Скрипт, прикрепленный к странице:
var Clone1=Vector1.CloneObject(100, 100) //клонировать векторное изображение
wait(5);Clone1.life=0 //обнулить пользовательское свойство у клона через 5 секунд
//Скрипт, прикрепленный к объекту Vector1:
this.life=1 //всем клонам векторного изображения присваивается данное пользовательское свойство
while(this.life==1){this.Spin(360, 1)} //пока пользовательское свойство не обнулилось, клон вертится
this.DestroyClonedObject() //уничтожить клон при обнулении пользовательского свойства
```

GetPersistentObject

Создать уникальную постоянную копию графического объекта (**???**).
Синтаксис:
GetPersistentObject()
Возврат:
(О), уникальная постоянная копия гр. объекта.
Пример:

```
var Persist1=Vector1.GetPersistentObject()
```

IsObjectIntersecting

Проверить, пересекаются ли на текущей странице два графических объекта (их ограничительные рамки) друг с другом.
Особенности функции:
1) Для проверки пересечения конкретных полигонов у векторных изображений или выступающих частей у растровых изображений необходимо: в свойствах объекта **"Properties"** на вкладке **"General"** в поле **"Options"** поставить флажок **"Ignore Transparent Area"**.
2) Для проверки столкновения фигурных растровых изображений следует использовать картинки в формате **.PNG** без альфа-канала, где неиспользуемый цвет (чаще всего **RGB(255, 0, 255)**, цвет **magenta**) будет указан как прозрачная область.
3) Часто поворот или скос гр. объекта (его ограничительной рамки) в векторном редакторе программы приводит к вылету, либо к **неправильной** работе функции и искаженному отображению самого гр. объекта в запущенной публикации. Это один из крупных багов программы, поэтому рекомендуется не поворачивать и не перекашивать гр. объекты средствами редактора, а производить эти операции с помощью скриптовых команд.
Синтаксис:
FirstObject.IsObjectIntersecting(SecondObject)
Возврат:
(Б), результат проверки на пересечение (**true** - есть пересечение, **false** - нет пересечения).
Параметры:
FirstObject - **(О)** исходный, к которому применяется функция.
SecondObject - **(О)** для проверки пересечения с исходным. **ОП**.
Пример:

```
Debug.trace(Vector1.IsObjectIntersecting(Vector2)) //проверка пересечения двух векторных объектов
```

GetWidth

Получить ширину ограничительной рамки графического объекта.

Синтаксис:

GetWidth()

Возврат:

(Ц), ширина гр. объекта (в пикселях).

Пример:

```
Debug.trace("Ширина: "+Vector1.GetWidth())
```

GetHeight

Получить высоту ограничительной рамки графического объекта.

Синтаксис:

GetHeight()

Возврат:

(Ц), высота гр. объекта (в пикселях).

Пример:

```
Debug.trace("Высота: "+Vector1.GetHeight())
```

SetObjectSize

Установить ширину и высоту ограничительной рамки графического объекта.

Особенности функции:

1) Изменение высоты и ширины происходит из центра ограничительной рамки гр. объекта.

2) Изменение растровых изображений зависит от свойств гр. объекта **"Properties"** на вкладке **"Image"** в поле **"Display Options"**.

3) Содержимое векторных объектов растягивается и в ширину, и в высоту искажаясь, в соответствии с установленным размером.

4) Содержимое кадров не масштабируется и не искажается, изменяется только размер кадра (при уменьшении размера - остается только то, что попало в кадр, а при увеличении размера - увеличивается пространство внутри кадра).

Синтаксис:

SetObjectSize(Width, Height)

Параметры:

Width - **(Ц)**, ширина гр. объекта (в пикселях). **ОП**.

Height - **(Ц)**, высота гр. объекта (в пикселях). **ОП**.

Пример:

```
Vector1.SetObjectSize(128, 256)
```

GetObjectDimensions

Получить данные о координатах (относительно верхнего левого угла страницы) и размере графического объекта (его ограничительной рамки). Если данный гр. объект скошен или повернут, то координаты и размер берутся так, как если бы он находился в исходном состоянии.

Синтаксис:

GetObjectDimensions()

Возврат:

(О) со следующими свойствами:

width - **(Ц)**, ширина ограничительной рамки гр. объекта.

height - **(Ц)**, высота ограничительной рамки гр. объекта.

left - **(Ц)**, **X**-координата левой стороны ограничительной рамки гр. объекта.

right - **(Ц)**, **X**-координата правой стороны ограничительной рамки гр. объекта.

top - **(Ц)**, **Y**-координата верхней стороны ограничительной рамки гр. объекта.

bottom - **(Ц)**, **Y**-координата нижней стороны ограничительной рамки гр. объекта.

Пример:

```
var V1DIM=Vector1.GetObjectDimensions() //переменная содержит свойства графического объекта
Debug.trace("Ширина объекта: "+V1DIM.width+" Высота объекта: "+V1DIM.height)
Debug.trace("\n"+"Координаты верхнего левого угла: "+V1DIM.left+", "+V1DIM.top)
Debug.trace("\n"+"Координаты нижнего правого угла: "+V1DIM.right+", "+V1DIM.bottom)
```

SetObjectDimensions

Установить размеры графического объекта равными размерам другого графического объекта. Данная функция работает **неправильно**:

1) Если ширина и высота исходного гр. объекта не равны ширине и высоте того гр. объекта, из которого брались свойства, то визуально исходный гр. объект будет только растянут из исходной точки своего верхнего левого угла, по размеру того гр. объекта, из которого были взяты свойства. Если же проверить свойства измененного исходного гр. объекта функцией **GetObjectDimensions**, то ширина и высота останутся прежними, а исходные координаты сместятся так, как если бы исходный гр. объект стоял в центре измененного себя.

2) Если ширина и высота исходного гр. объекта равны ширине и высоте того гр. объекта, из которого брались свойства, то исходный гр. объект изменит свои координаты правильно, и переместится на то же место, где стоит гр. объект из которого брались свойства.

Синтаксис:

SetObjectDimensions(DimensionObject)

Параметры:

DimensionObject - **(О)** со всеми или несколькими свойствами (описаны у функции **GetObjectDimensions**) для применения к указанному гр. объекту. **ОП**.

Пример:

```
var V1DIM=Vector1.GetObjectDimensions() //переменная содержит свойства графического объекта
Vector2.SetObjectDimensions(V1DIM) //применить свойства к другому графическому объекту
```

GetDisplayData

Получить свойства трансформации графического объекта.

Особенности функции:

1) Начальные координаты любого гр. объекта, располагающегося на любой позиции считаются равными **(0, 0)**, они будут точкой отсчета для объекта.

2) Если кадр, содержащий гр. объект, изменил координаты на странице, поменял угол или был масштабирован, то значения координат, угла и масштаба гр. объекта, расположенного внутри, не изменятся.

3) Функция работает **неправильно** при взятии величин перекоса (**skewx** и **skewy**). Полученные значения умножаются на **10**. Чтобы правильно передать свойства от одного гр. объекта к другому, необходимо предварительно поделить значения этих двух свойств на **10**.

Синтаксис:

GetDisplayData()

Возврат:

(О) со следующими свойствами:

x - **(Ц)**, **X**-координата центра гр. объекта, относительно его начальной позиции.

y - **(Ц)**, **Y**-координата центра гр. объекта, относительно его начальной позиции.

scalex - **(Ч)**, текущая величина масштабирования гр. объекта в **X**-направлении.

scaley - **(Ч)**, текущая величина масштабирования гр. объекта в **Y**-направлении.

angle - **(Ч)**, текущий угол поворота в градусах гр. объекта вокруг центра.

anglex - **(Ч)**, текущий угол поворота в градусах гр. объекта вокруг **X**-оси в **3D**.

angley - **(Ч)**, текущий угол поворота в градусах гр. объекта вокруг **Y**-оси в **3D**.

skewx - **(Ч)**, текущая величина перекоса гр. объекта в **X**-направлении.

skewy - **(Ч)**, текущая величина перекоса гр. объекта в **Y**-направлении.

transparency - **(Ц)**, текущий уровень прозрачности гр. объекта (**0** - прозрачность отсутствует, **100** - полная прозрачность).

SetDisplayData

Установить свойства трансформации для графического объекта.

Особенности функции:

1) Начальные координаты любого гр. объекта, располагающегося на любой позиции считаются равными **(0, 0)**, они будут его точкой отсчета.

2) Если кадр, содержащий гр. объект, изменил координаты на странице, поменял угол или был масштабирован, то значения координат, угла и масштаба гр. объекта, расположенного внутри, не изменятся.

Синтаксис:

SetDisplayData(PositionObject)

Параметры:

PositionObject - **(О)** со всеми или несколькими свойствами (описаны у функции **GetDisplayData**) для применения к гр. объекту. **ОП**.

Общий пример:

```
var ObjInfo=new Object() //создать объект для хранения свойств
ObjInfo.x=50;ObjInfo.y=50;ObjInfo.angle=45.5;ObjInfo.anglex=45.5;ObjInfo.angley=45.5
ObjInfo.scalex=50.5;ObjInfo.scaley=50.5;ObjInfo.skewx=55.5;ObjInfo.skewy=55.5
ObjInfo.transparency=50 //присвоить значения свойствам
wait(3);Vector1.SetDisplayData(ObjInfo) //передать свойства векторному изображению
ObjInfo=Vector1.GetDisplayData() //взять свойства у одного векторного изображения
ObjInfo.skewx/=10;ObjInfo.skewy/=10 //исправить ошибочные значения
Debug.trace("Смещение по X-оси: "+ObjInfo.x+" Смещение по Y-оси: "+ ObjInfo.y+"\n")
Debug.trace("Масштабирование X: "+ObjInfo.scalex+" Масштабирование Y: "+ObjInfo.scaley+"\n")
Debug.trace("Перекос X: "+ObjInfo.skewx+" Перекос Y: "+ObjInfo.skewy+"\n")
Debug.trace("Прозрачность: "+ObjInfo.transparency+" Угол поворота: "+ObjInfo.angle+"\n")
Debug.trace("Поворот вокруг X-оси: "+ObjInfo.anglex+" Поворот вокруг Y-оси: "+ObjInfo.angley)
wait(3);Vector2.SetDisplayData(ObjInfo) //передать свойства другому векторному изображению
```

GetPosition, GetXPosition и GetYPosition

Получить координаты графического объекта. Отсчет координат при использовании этих функций **всегда** происходит относительно верхнего левого угла страницы, даже если гр. объект находится внутри кадра. Следует внимательно использовать эти функции вместе с **SetPosition**, **SetPositionX**, **SetPositionY**, из-за различий отсчета координат для гр. объекта внутри кадра.

Синтаксис:

GetPosition()
GetXPosition()
GetYPosition()

Возврат:

GetPosition возвращает:
(О) со следующими свойствами:
x - **(Ч)**, **X**-координата центра гр. объекта относительно верхнего левого угла страницы.
y - **(Ч)**, **Y**-координата центра гр. объекта относительно верхнего левого угла страницы.
GetXPosition возвращает:
(Ч), **X**-координата центра гр. объекта относительно верхнего левого угла страницы.
GetYPosition возвращает:
(Ч), **Y**-координата центра гр. объекта относительно верхнего левого угла страницы.

Пример:

```
var ObjPos=Vector1.GetPosition() //переменная содержит объект с координатами изображения
Debug.trace ("X: "+ObjPos.x+" Y: "+ObjPos.y+"\n")
XObjPos=Vector1.GetXPosition() //переменная содержит значение только X-координаты изображения
YObjPos=Vector1.GetYPosition() //переменная содержит значение только Y-координаты изображения
Debug.trace ("X: "+XObjPos+" Y: "+YObjPos)
```

SetPosition, SetPositionX, SetPositionY и Move, MoveX, MoveY

Изменить координаты графического объекта.
Если гр. объект располагается на странице внутри кадра, то координаты отсчитываются от верхнего левого угла кадра, а если располагается на странице и не принадлежит кадру, то координаты отсчитываются от верхнего левого угла страницы.
Если кадр, содержащий гр. объект, повернут на угол (или перекошен), то и оси координат внутри кадра повернутся на тот же угол (или перекосятся), и если гр. объект внутри этого кадра соответственно координатам переместится прямолинейно, то выглядеть перемещение гр. объекта относительно страницы будет угловым или скошенным.

Синтаксис:

SetPosition(PosX, PosY, Time, Wait)
SetPositionX(PosX, Time, Wait)
SetPositionY(PosY, Time, Wait)
Move(PosX, PosY, Time, Wait)
MoveX(PosX, Time, Wait)
MoveY(PosY, Time, Wait)

Параметры:

PosX - **(Ц)** (может быть отрицательным), величина перемещения центра гр. объекта по **X**-оси координат. **SetPosition** и **SetPositionX** изменят текущую **X**-координату гр. объекта, на ту, что указана в параметре, а **Move** и **MoveX** переместят гр. объект по **X**-оси координат на столько пикселей от текущей позиции, сколько указано в параметре. **ОП**.
PosY - **(Ц)** (может быть отрицательным), величина перемещения центра гр. объекта по **Y**-оси координат. **SetPosition** и **SetPositionY** изменят текущую **Y**-координату гр. объекта, на ту, что указана в параметре, а **Move** и **MoveY** переместят гр. объект по **Y**-оси координат на столько пикселей от текущей позиции, сколько указано в параметре. **ОП**.
Time - **(Ч)**, продолжительность анимации действия в секундах. **НП**, по умолчанию **0.00**.
Wait - **(Б)**, выполнение следующей строчки кода (**true** - только после завершения анимации действия, **false** - сразу после старта анимации действия). **НП**, по умолчанию **true**.

Пример:

```
Vector1.SetPosition(100, 100, 3.5) //переместить изображение в координаты 100,100 за 3.5 секунды
Vector1.Move(100, 100, 3.5) //переместить изображение по диагонали вправо-вниз за 3.5 секунды
```

GetPosition и GetPositionFromPercent

Получить координаты заданной точки на векторной линии анимационного пути.

Синтаксис:

GetPosition(Length)

GetPositionFromPercent(Percent)

Возврат:

(О) со следующими свойствами:

x - **(Ц)**, **X**-координата заданной точки на векторной линии указанного анимационного пути, относительно центра его ограничительной рамки.

y - **(Ц)**, **Y**-координата заданной точки на векторной линии указанного анимационного пути, относительно центра его ограничительной рамки.

Параметры:

Length - **(Ц)**, пиксельное расстояние от начала и вдоль векторной линии анимационного пути до точки, проверяемой на координаты. **ОП**.

Percent - **(Ч)**, процентное расстояние от начала и вдоль векторной линии анимационного пути до точки, проверяемой на координаты. **ОП**.

Пример:

```
var Path1Pos=Path1.GetPosition(Path1.GetTotalLength()/2)
var Path1Perc=Path1.GetPositionFromPercent(100)
Debug.trace("X середины пути: "+Path1Pos.x+"\n"+"Y середины пути: "+Path1Pos.y+"\n")
Debug.trace("X конца пути: "+Path1Perc.x+"\n"+"Y конца пути: "+Path1Perc.y)
```

GetTotalLength

Получить длину векторной линии анимационного пути.

Синтаксис:

GetTotalLength()

Возврат:

(Ч), длина (в пикселях) векторной линии анимационного пути.

Пример:

```
Debug.trace("Длина пути в пикселях: "+Path1.GetTotalLength())
```

FollowPath

Анимировать графический объект вдоль анимационного пути.

Синтаксис:

FollowPath(Path, Time, Relative, Orientation, Start, End, Wait)

Параметры:

Path - **(О)**, анимационный путь следования гр. объекта. **ОП**.

Time - **(Ч)**, количество секунд, которое требуется при следовании по пути. **НП**, по умолчанию **0.0** секунд.

Relative - **(Б)**, движение гр. объекта вдоль пути (**true** - относительно текущей позиции гр. объекта, **false** - со следованием по траектории абсолютно). **НП**, по умолчанию **true**.

Orientation - **(Б)**, ориентация гр. объекта вдоль пути (**true** - выровненная по траектории, **false** - фиксированная). **НП**, по умолчанию **false**.

Start - **(Ч)**, процент от **0** до **100**, начальная позиция для гр. объекта вдоль пути. **НП**, по умолчанию **0**.

End - **(Ч)**, процент от **0** до **100**, конечная позиция для гр. объекта вдоль пути. **НП**, по умолчанию **100**.

Wait - **(Б)**, выполнение следующей строчки кода (**true** - только после завершения анимации действия, **false** - сразу после старта анимации действия). **НП**, по умолчанию **true**.

Пример:

```
Vector1.FollowPath(Path1, 2.0, false, true, 10, 90, true)
```

SetTransparency и Fade

Установить прозрачность графического объекта.

Синтаксис:

SetTransparency(Trans, Time, Wait)
Fade(Trans, Time, Wait)

Параметры:

Trans - **(Ц)**, процент прозрачности гр. объекта (**0** - полная непрозрачность, **100** - полная прозрачность). **SetTransparency** поменяет текущее значение процента прозрачности гр. объекта, на такое, какое указано в параметре, а **Fade** увеличит/уменьшит текущий процент прозрачности гр. объекта на столько процентов, сколько указано в параметре. Отрицательные значения используются только для **Fade** при уменьшении процента прозрачности. **ОП**.

Time - **(Ч)**, продолжительность анимации действия в секундах. **НП**, по умолчанию **0.00**.

Wait - **(Б)**, выполнение следующей строчки кода (**true** - только после завершения анимации действия, **false** - сразу после старта анимации действия). **НП**, по умолчанию **true**.

Пример:

```
//исходное изображение должно быть полностью непрозрачно
Vector1.SetTransparency(90, 10) //изображение станет прозрачным на 90% за 10 секунд
Vector1.Fade(-80, 10) //прозрачность изображения за 10 секунд снизится на 80% до значения 10%
```

SetRotation и Rotate

Повернуть графический объект на нужный угол вокруг точки вращения, расположенной в центре объекта.

Синтаксис:

SetRotation(Angle, Direction, Time, Wait)
Rotate(Angle, Time, Wait)

Параметры:

Angle - **(Ч)**, угол поворота графического объекта в градусах. **SetRotation** изменит текущий угол поворота графического объекта, на тот, что указан в параметре, а **Rotate** добавит к текущему углу поворота гр. объекта столько градусов, сколько указано в параметре. Отрицательные значения используются только для **Rotate**, чтобы направление вращения было против часовой стрелки. **ОП**.

Direction - **(Б)**, направление вращения указанного гр. объекта (**true** - по часовой стрелке, **false** - против часовой стрелки). **ОП**.

Time - **(Ч)**, продолжительность анимации действия в секундах. **НП**, по умолчанию **0.00**.

Wait - **(Б)**, выполнение следующей строчки кода (**true** - только после завершения анимации действия, **false** - сразу после старта анимации действия). **НП**, по умолчанию **true**.

Пример:

```
//исходное изображения не должно быть повернуто
Vector1.SetRotation(360, true, 10) //изображение совершит один полный поворот по часовой стрелке
Vector1.Rotate(-360, 10) //изображение совершит один полный поворот против часовой стрелки
```

SetRoll и Roll

Повернуть графический объект на нужный угол вокруг **X**-оси, проходящей через середину объекта.

Синтаксис:

SetRoll(Angle, Direction, Time, Wait)
Roll(Angle, Time, Wait)

Параметры:

Angle - **(Ч)**, угол поворота графического объекта в градусах. **SetRoll** изменит текущий угол поворота графического объекта, на тот, что указан в параметре, а **Roll** добавит к текущему углу поворота графического объекта столько градусов, сколько указано в параметре. Отрицательные значения используются только для **Roll**, чтобы направление вращения было назад. **ОП**.

Direction - **(Б)**, направление вращения графического объекта (**true** - вперёд, **false** - назад). **ОП**.

Time - **(Ч)**, продолжительность анимации действия в секундах. **НП**, по умолчанию **0.00**.

Wait - **(Б)**, выполнение следующей строчки кода (**true** - только после завершения анимации действия, **false** - сразу после старта анимации действия). **НП**, по умолчанию **true**.

Пример:

```
//исходное изображения не должно быть повернуто
Vector1.SetRoll(180, false, 20) //изображение перевернется сверху вниз, направление вращения назад
Vector1.Roll(180, 20) //изображение перевернется сверху вниз, направление вращения вперёд
```

SetSpin и Spin

Повернуть графический объект на нужный угол вокруг **Y**-оси, проходящей через середину объекта.

Синтаксис:

SetSpin(Angle, Direction, Time, Wait)
Spin(Angle, Time, Wait)

Параметры:

Angle - **(Ч)**, угол поворота графического объекта в градусах. **SetSpin** изменит текущий угол поворота графического объекта, на тот, что указан в параметре, а **Spin** добавит к текущему углу поворота графического объекта столько градусов, сколько указано в параметре. Отрицательные значения используются только для **Spin**, чтобы направление вращения было налево. **ОП**.
Direction - **(Б)**, направление вращения графического объекта (**true** - направо, **false** - налево). **ОП**.
Time - **(Ч)**, продолжительность анимации действия в секундах. **НП**, по умолчанию **0.00**.
Wait - **(Б)**, выполнение следующей строчки кода (**true** - только после завершения анимации действия, **false** - сразу после старта анимации действия). **НП**, по умолчанию **true**.

Пример:

```
//исходное изображения не должно быть повернуто
Vector1.SetSpin(315, false, 3) //изображение повернется налево
Vector1.Spin(90, 6) //изображение повернется направо
```

SetScale, SetScaleH, SetScaleV и Scale, ScaleH, ScaleV

Масштабировать графический объект. Масштабирование происходит из центра объекта.
Изначальный масштаб графического объекта на странице всегда **100%**.

Синтаксис:

SetScale(Horizontal, Vertical, Time, Wait)
SetScaleH(Horizontal, Time, Wait)
SetScaleV(Vertical, Time, Wait)
Scale(Horizontal, Vertical, Time, Wait)
ScaleH(Horizontal, Time, Wait)
ScaleV(Vertical, Time, Wait)

Параметры:

Horizontal - **(Ч)**, горизонтальный коэффициент масштабирования графического объекта (например **1.5** - это масштаб **150%**, а **0.25** - это масштаб **25%**). **SetScale** и **SetScaleH** изменят текущий горизонтальный коэффициент масштабирования графического объекта, на тот, что указан в параметре, а **Scale** и **ScaleH** добавят к текущему горизонтальному коэффициенту масштабирования графического объекта столько, сколько указано в параметре. Отрицательные значения используются только для **Scale** и **ScaleH**, чтобы уменьшать масштаб по горизонтали. **ОП**.
Vertical - **(Ч)**, вертикальный коэффициент масштабирования графического объекта. Действует по подобию горизонтального. **ОП**.
Time - **(Ч)**, продолжительность анимации действия в секундах. **НП**, по умолчанию **0.00**.
Wait - **(Б)**, выполнение следующей строчки кода (**true** - только после завершения анимации действия, **false** - сразу после старта анимации действия). **НП**, по умолчанию **true**.

Пример:

```
Vector1.SetScale(4, 4, 10) //масштаб изображения из 100% станет 400% за 10 секунд
Vector1.Scale(-3.5, -3.5, 10) //масштаб изображения из 400% станет 50% за 10 секунд
```

SetSkew, SetSkewH, SetSkewV и Skew, SkewH, SkewV

Перекосить графический объект.

Синтаксис:

SetSkew(Horizontal, Vertical, Time, Wait)
SetSkewH(Horizontal, Time, Wait)
SetSkewV(Vertical, Time, Wait)
Skew(Horizontal, Vertical, Time, Wait)
SkewH(Horizontal, Time, Wait)
SkewV(Vertical, Time, Wait)

Параметры:

Horizontal - **(Ч)**, процент горизонтального перекоса графического объекта (от **-200** до **200**, где **0** - отсутствие перекоса). **SetSkew** и **SetSkewH** поменяют текущее значение процента горизонтального перекоса графического объекта, на такое, какое указано в параметре, а **Skew** и **SkewH** увеличат/уменьшат текущий процент горизонтального перекоса графического объекта на столько процентов, сколько указано в параметре. **ОП**.

Vertical - **(Ч)**, процент вертикального перекоса для графического объекта. Действует по подобию горизонтального. **ОП**.

Time - **(Ч)**, продолжительность анимации действия в секундах. **НП**, по умолчанию **0.00**.

ВАЖНО: Функция **SetSkewV** (ей лучше вообще не пользоваться) - единственная, что **работает с ошибкой** и **не реагирует** на значение **0** (требуется хотя бы **0.01**), а также иногда приводит к вылету программы.

Wait - **(Б)**, выполнение следующей строчки кода (**true** - только после завершения анимации действия, **false** - сразу после старта анимации действия). **НП**, по умолчанию **true**.

Пример:

```
Vector1.SetSkew(-200, -200, 10) //перекосить объект по горизонтали и вертикали за 10 секунд
Vector1.Skew(200, 200, 10) //изображение через 10 секунд перестанет быть перекошенным
```

StopAnimation и ResetAnimation

Остановить/сбросить выбранные анимации действий для графического объекта.

Синтаксис:

StopAnimation(StopCode, Wait)
ResetAnimation(StopCode, Time, Wait)

Параметры:

StopCode - **(С)**, один или несколько кодов анимации для остановки/сброса, разделенных символом вертикальной черты [|]. **ОП**.

Коды анимации:

"Roll" - вращение вокруг **X**-оси.
"Spin" - вращение вокруг **Y**-оси.
"Rotate" - вращение вокруг центра.
"Scale" - горизонтальное и вертикальное масштабирование.
"ScaleH" - горизонтальное масштабирование.
"ScaleV" - вертикальное масштабирование.
"Skew" - горизонтальное и вертикальное перекашивание.
"SkewH" - горизонтальное перекашивание.
"SkewV" - вертикальное перекашивание.
"Move" - горизонтальное и вертикальное движение.
"MoveX" - горизонтальное движение.
"MoveY" - вертикальное движение.
"Fade" - изменение прозрачности.
"Path" - следование по анимационному пути.
"ALL" - все текущие анимации.

Wait - **(Б)**, выполнение следующей строчки кода (**true** - только после завершения сброса/остановки, **false** - сразу после старта сброса/остановки). **НП**, по умолчанию **true**.

Time - **(Ч)**, время в секундах, пауза перед сбросом/остановкой выбранных анимаций. **НП**, по умолчанию **0.00**.

Пример:

```
Vector1.StopAnimation("Roll|Spin|ScaleH|Skew")
Vector1.ResetAnimation("Roll|Spin", 1.0)
```

GetScrollInfo

Получить информацию о состоянии прокрутки графического объекта. Для прокрутки следует в свойствах графического объекта **"Properties"** на вкладке **"Image"** в поле **"Display Options"** выбрать опцию **"Fixed"**.

Синтаксис:

GetScrollInfo()

Возврат:

(О) со следующими свойствами:

vscrollbar - **(О)**, вертикальная полоса прокрутки с ползунком, если существует, иначе **undefined**.

vvisible - **(Ч)**, процент от настоящей высоты гр. объекта.

vpos - **(Ч)**, процентная позиция вертикальной прокрутки.

hscrollbar - **(О)**, горизонтальная полоса прокрутки с ползунком, если существует, иначе **undefined**.

hvisible - **(Ч)**, процент от настоящей ширины гр. объекта.

hpos - **(Ч)**, процентная позиция горизонтальной прокрутки.

Пример:

```
var Sc1=Image1.GetScrollInfo()
Debug.trace("Позиция вертикальной прокрутки (%): "+Sc1.vpos+"\n")
Debug.trace("Позиция горизонтальной прокрутки (%): "+Sc1.hpos+"\n")
```

SetScrollPosition

Изменить позицию прокрутки графического объекта.

Синтаксис:

SetScrollPosition(VScrollPos, HScrollPos)

Параметры:

VScrollPos - **(Ч)**, процентная позиция вертикальной прокрутки, **-1** чтобы не изменять. **ОП**.

HScrollPos - **(Ч)**, процентная позиция горизонтальной прокрутки, **-1** чтобы не изменять. **ОП**.

Пример:

```
Image1.SetScrollPosition(50, 50) //прокрутка в центр
```

GetAppearance

Получить объект внешнего вида для графического объекта в определенном состоянии (нельзя применить к другому гр. объекту). К объекту внешнего вида можно применить следующие функции: **SetImage**, **RemoveImage**, **SetBackground**, **RemoveBackground**, **SetTexture**, **RemoveTexture**, **SetAlpha**, **RemoveAlpha**, **SetShadow**, **RemoveShadow**, **SetFlare**, **RemoveFlare**, **SetBorder**, **RemoveBorder**, **SetBtnColour**, **RemoveBtnColour**.

Синтаксис:

GetAppearance(Appearance)

Возврат:

(О) со свойствами внешнего вида гр. объекта в выбранном состоянии.

Параметры:

Appearance - **(С)**, название состояния гр. объекта. **НП**, по умолчанию **"Default"**.

Возможные состояния:

"Default" - обычное.

"Mouse Over" - над гр. объектом находится курсор мыши.

"Mouse Pressed" - на гр. объект нажата левая кнопка мыши.

"Disabled" - гр. объект выключен.

Пример:

```
State1=Frame1.GetAppearance("Default");State2=Frame1.GetAppearance("Mouse Over")
State3=Frame1.GetAppearance("Mouse Pressed");State4=Frame1.GetAppearance("Disabled")
```

ReloadImage

Перезагрузить текущее фоновое растровое изображение для графического объекта (**???**).

Синтаксис:

ReloadImage()

Пример:

```
Frame1.ReloadImage()
```

SetBackground

Создать цветной фон для графического объекта. Цветной фон находится позади фонового растрового изображения.

Синтаксис:

SetBackground(PropertyObj)

Параметры:

PropertyObj - **(О)**, содержащий выбранные свойства фона. **ОП**.

Свойство стиля фона:

style - **(С)**, стиль фона, возможные значения:

"solid" - сплошная одноцветная заливка.

"linear", **"centre"**, **"circular"**, **"radial"**, **"spiral"**, **"spiral2"**, **"exponentialspiral"**, **"concentric"** - различные варианты градиентной заливки.

Свойства для сплошного и градиентных фонов:

colour - **(Ц)** или **(С)**, **RGB**-цвет фона. Для сплошного фона используется одно значение цвета, а для градиентного массив.

transparency - **(Ц)** от **0** (полная прозрачность) до **100** (полная непрозрачность), прозрачность фона.

Свойства только для градиентных фонов:

angle - **(Ц)** от **0** до **359**, угол направления градиента в градусах (только для стилей **"linear"**, **"radial"**, **"spiral"**, **"spiral2"**, **"exponentialspiral"**).

twist - **(Ч)** от **0** до **2** (возможны значительно большие значения), закрученность завитка (только для стилей **"spiral"**, **"spiral2"**, **"exponentialspiral"**).

startsize - **(Ч)** от **1** до **100**, процентное смещение полос (только для стилей **"spiral"**, **"spiral2"**, **"exponentialspiral"**, **"concentric"**).

bands - **(Ц)** от **0** до **20** (возможны значительно большие значения), количество полос (только для стилей **"spiral"**, **"spiral2"**, **"exponentialspiral"**, **"concentric"**).

offsetx - **(Ц)** от **0** до **100**, процентное горизонтальное смещение центра градиента от левого верхнего угла (только для стилей **"centre"**, **"circular"**, **"radial"**, **"spiral"**, **"spiral2"**, **"exponentialspiral"**, **"concentric"**).

offsety - **(Ц)** от **0** до **100**, процентное вертикальное смещение центра градиента от левого верхнего угла (только для стилей **"centre"**, **"circular"**, **"radial"**, **"spiral"**, **"spiral2"**, **"exponentialspiral"**, **"concentric"**).

Пример:

```
var BGStyle1=new Object()
BGStyle1.style="spiral"
BGStyle1.colour=new Array()
BGStyle1.colour[0]=RGB(128, 0, 0)
BGStyle1.colour[1]=RGB(0, 128, 0)
BGStyle1.colour[2]=RGB(0, 0, 128)
BGStyle1.twist=0.3
BGStyle1.bands=3
BGStyle1.offsetx=50
BGStyle1.offsety=50
Frame1.SetBackground(BGStyle1)
```

RemoveBackground

Убрать текущий цветной фон для графического объекта.

Синтаксис:

RemoveBackground()

Пример:

```
Frame1.RemoveBackground()
```

SetImage

Установить фоновое растровое изображение для графического объекта (цвет прозрачности сбрасывается).

Синтаксис:

SetImage(Filename)

Параметры:

Filename - **(С)**, путь к файлу изображения. **ОП**.

Пример:

```
Frame1.SetImage(SYSTEM_PUBLICATION_DIR+"\\Picture.png")
```

RemoveImage

Убрать текущее фоновое растровое изображение для графического объекта.

Синтаксис:

RemoveImage()

Пример:

```
Frame1.RemoveImage()
```

SetTexture

Создать эффект текстуры для графического объекта. Для анимированных текстур используются **GIF** -изображения.

Синтаксис:

SetTexture(Gloss, Invert, Animated, ImagePath, Tile)

Параметры:

Gloss - **(Ц)** от **0** до **20** или **(С)** (**"Paper"**, **"Card"**, **"Silk"**, **"Plastic"**, **"Metal"**, **"Glass"**), стиль блеска. **НП**, по умолчанию **"Plastic"**.

Invert - **(Б)**, инвертирование текстуры (**true** - включить, **false** - выключить). **НП**, по умолчанию **false**.

Animated - **(Б)**, анимация текстуры если есть (**true** - включить, **false** - выключить). **НП**, по умолчанию **false**.

ImagePath - **(С)**, путь к файлу изображения. **НП**, по умолчанию текстура, назначенная в редакторе.

Tile - **(Б)**, тип текстуры (**true** - мозаичная, **false** - растянутая). **НП**, по умолчанию **false**.

Пример:

```
Texture1Path=SYSTEM_PUBLICATION_DIR+"\\Texture.png"
Vector1.SetTexture("Silk", false, false, Texture1Path, false)
```

RemoveTexture

Убрать текущий эффект текстуры для графического объекта.

Синтаксис:

RemoveTexture()

Пример:

```
Vector1.RemoveTexture()
```

SetAlpha

Создать альфа-эффект (градиент прозрачности) для графического объекта.

Синтаксис:

SetAlpha(Style, Transp1, Transp2)

Параметры:

Style - **(С)**, название стиля альфа-эффекта. **ОП**.

Возможные значения: **"None"**, **"Horizontal"**, **"Vertical"**, **"NWSE"**, **"NESW"**, **"Centre"**, **"Circle"**.

Transp1 - **(Ц)** от **0** до **100**, процент минимальной прозрачности альфа-канала. **НП**, по умолчанию **0**.

Transp2 - **(Ц)** от **0** до **100**, процент максимальной прозрачности альфа-канала. **НП**, по умолчанию **100**.

Пример:

```
Vector1.SetAlpha("Horizontal", 0, 100)
```

RemoveAlpha

Убрать текущий альфа-эффект для графического объекта.

Синтаксис:

RemoveAlpha()

Пример:

```
Vector1.RemoveAlpha()
```

SetShadow

Создать эффект тени для графического объекта.

Синтаксис:

SetShadow(Height, Width, Transparency, Colour, Blur)

Параметры:

Height - **(Ц)** положительное, высота тени в пикселях. **НП**, по умолчанию **10**.

Width - **(Ц)** положительное, ширина тени в пикселях. **НП**, по умолчанию **10**.

Transparency - **(Ц)** от **0** до **100**, процент прозрачности тени (**0** - прозрачность отсутствует, **100** - полная прозрачность). **НП**, по умолчанию **75**.

Colour - **(Ц)** или **(С)**, цвет тени. **НП**, по умолчанию **0** (чёрный).

Blur - **(Ц)** от **0** до **5**, размытие тени в пикселях. **НП**, по умолчанию **2**.

Пример:

Vector1.SetShadow(5, 3, 50, RGB(128, 64, 128), 3)

RemoveShadow

Убрать текущий эффект тени для графического объекта.

Синтаксис:

RemoveShadow()

Пример:

Vector1.RemoveShadow()

SetFlare

Создать вокруг графического объекта эффект свечения нужного цвета.

Синтаксис:

SetFlare(Height, Width, Transparency, Colour, Blur)

Параметры:

Height - **(Ц)** от **1** до **30**, высота свечения в пикселях. **НП**, по умолчанию **3**.

Width - **(Ц)** от **1** до **30**, ширина свечения в пикселях. **НП**, по умолчанию **3**.

Можно выставить (но не рекомендуется) значения высоты и ширины свечения больше **30**, которые подействуют. Чем больше эти значения превышают норму, тем медленнее будет работать публикация, а слишком большие значения приведут к вылету.

Transparency - **(Ц)** от **0** до **100**, процент прозрачности свечения (**0** - прозрачность отсутствует, **100** - полная прозрачность). **НП**, по умолчанию **50**.

Colour - **(Ц)** или **(С)**, цвет свечения. **НП**, по умолчанию **0** (чёрный).

Blur - **(Ц)** от **0** до **5**, значение размытия в пикселях для блика. **НП**, по умолчанию **3**.

Пример:

Vector1.SetFlare(5, 5, 50, RGB(80, 160, 48), 5)

RemoveFlare

Убрать текущий эффект свечения для графического объекта.

Синтаксис:

RemoveFlare()

Пример:

Vector1.RemoveFlare()

SetBorder

Создать эффект рамки для графического объекта.

Синтаксис:

SetBorder(Style, Round, Width, Radius, Colour1, Colour2, Opacity)

Параметры:

Style - **(С)**, стиль рамки. **ОП**.

Возможные значения:

"None" - без рамки;

"Mask" - назначенный в редакторе программы;

"Plain", **"Ellipse"** - одноцветные;

"Double", **"Neon"**, **"Raised"**, **"Sunken"** - двухцветные.

Round - **(Б)**, закругленность углов рамки (**true** - да, **false** - нет). **НП**, по умолчанию **false**.

Width - **(Ц)**, ширина рамки в пикселях. **НП**, по умолчанию **0**.

Radius - **(Ц)**, радиус закругления угла рамки в пикселях. **НП**, по умолчанию **0**.

Colour1 - **(Ц)** или **(С)**, первый цвет рамки. **НП**, по умолчанию **0** (чёрный).

Colour2 - **(Ц)** или **(С)**, второй цвет рамки. **НП**, по умолчанию **0** (чёрный).

Opacity - **(Ц)** от **0** до **100**, процент непрозрачности рамки (**0** - полная прозрачность, **100** - полная непрозрачность). Только для стилей **"Raised"** и **"Sunken"**. **НП**, по умолчанию **0**.

Пример:

```
Vector1.SetBorder("Plain", true, 5, 15, RGB(0, 0, 255))
```

RemoveBorder

Убрать текущий эффект рамки для графического объекта.

Синтаксис:

RemoveBorder()

Пример:

```
Vector1.RemoveBorder()
```

SetBtnColour

Создать эффект кнопки для графического объекта.

Синтаксис:

SetBtnColour(SurfaceCol, LightBevelCol, DarkBevelCol, BevelWidth, BevelOpacity, BlackOutline)

Параметры:

SurfaceCol - **(Ц)** или **(С)**, цвет поверхности кнопки. **НП**, по умолчанию серый.

LightBevelCol - **(Ц)** или **(С)**, цвет светлого скоса кнопки. **НП**, по умолчанию белый.

DarkBevelCol - **(Ц)** или **(С)**, цвет темного скоса кнопки. **НП**, по умолчанию чёрный.

BevelWidth - **(Ц)**, ширина скоса кнопки в пикселях. **НП**, по умолчанию 4.

BevelOpacity - **(Ц)** от **0** до **100**, процент непрозрачности скоса (**0** - полная прозрачность, **100** - полная непрозрачность). **НП**, по умолчанию **75**.

BlackOutline - **(Б)**, чёрный контур вокруг кнопки (**true** - с контуром, **false** - без контура). **НП**, по умолчанию **true**.

Пример:

```
CLR1=RGB(84, 62, 134)
CLR2=RGB(104, 84, 148)
CLR3=RGB(52, 14, 114)
Vector1.SetBtnColour(CLR1, CLR2, CLR3, 10, 100, false)
```

RemoveBtnColour

Убрать текущий эффект кнопки для графического объекта.

Синтаксис:

RemoveBtnColour()

Пример:

```
Vector1.RemoveBtnColour()
```

SetFocus

Сфокусировать весь ввод с клавиатуры на графическом объекте.

Синтаксис:

SetFocus()

Пример:

TextInput1.SetFocus()

ClearInputFocus

Отменить фокусировку ввода с клавиатуры на графическом объекте.

Синтаксис:

ClearInputFocus()

Пример:

TextInput1.ClearInputFocus()

CaptureMouse

Сфокусировать весь ввод мышью на графическом объекте.

Синтаксис:

CaptureMouse()

Пример:

Button1.CaptureMouse()

ReleaseMouse

Отменить фокусировку ввода мышью на графическом объекте.

Синтаксис:

ReleaseMouse()

Пример:

Button1.ReleaseMouse()

CopyToClipboard

Скопировать графический объект в буфер обмена.

Особенности функции:

1) Прозрачные части копируемого изображения после вставки из буфера обмена будут чёрного цвета.

2) Если копировать текстовый объект с русскими буквами как текст с форматированием, то при вставке будут кракозябры.

Синтаксис:

CopyToClipboard(Native)

Параметры:

Native - **(Б)**, тип копируемых данных только для текстовых графических объектов (**true** - текст с форматированием, **false** - текст как изображение). **НП**, по умолчанию **false**.

Пример:

Image1.CopyToClipboard()

PrintObject

Распечатать графический объект.

Синтаксис:

PrintObject(PrintDialog, CancelDialog)

Параметры:

PrintDialog - **(Б)**, отображение окна свойств печати перед распечаткой (**true** - окно появится, **false** - нет). **НП**, по умолчанию **false**.

CancelDialog - **(Б)**, отображение окна отмены печати во время распечатки (**true** - окно появится, **false** - нет). **НП**, по умолчанию **true**.

Пример:

Image1.PrintObject()

Текстовые объекты

Функции текстовых объектов предназначены для управления текстом на странице публикации.

Быстрый способ отслеживания текущего значения переменной с помощью текстового объекта:
1) Переменная обязательно должна быть объявлена в самом редакторе:
пункт меню **"Edit"** или **"Publication"** > диалоговое окно **"Publication Properties"** > вкладка **"Variables"**.
2) Для вставки переменой в текстовый объект: пункт меню **"Text"** > строка меню **"Insert Variable"**.
В текстовом объекте переменная будет заключена в угловые скобки (шевроны) [<>].
3) Если данная переменная будет переобъявлена в скрипте с помощью **var**, то её отслеживание в текстовом объекте прекратится.

Список функций:

GetParagraphCount - получить количество абзацев в текстовом объекте.
GetLineCount - получить количество строчек в текстовом объекте.
GetWordCount - получить количество слов в текстовом объекте.
GetTextLength - получить количество символов в текстовом объекте.
ParagraphLength - получить количество символов в абзаце.
LineLength - получить количество символов в строчке.
ParagraphIndex - получить номер индексной позиции первого символа нужного абзаца.
LineIndex - получить номер индексной позиции первого символа нужной строчки.
LineFromChar - получить номер строчки по номеру индексной позиции символа, расположенного в ней.
PointFromChar - получить координаты и размер символа по номеру его индексной позиции.
CharFromPoint - получить номер индексной позиции символа по его координатам.
SetSelection - выбрать фрагмент текста в текстовом объекте.
SelectParagraph - выбрать текст указанного абзаца в текстовом объекте.
ReplaceSelection - заменить текущий выбор указанной строкой.
GetSelectionText - получить выбранный текст в виде строки.
GetSelection - получить положение выбранных символов.
FindText - найти фрагмент текста в текстовом объекте.
FindTextInSelection - найти фрагмент текста только в выбранной части текстового объекта.
SetColour - установить цвет текста.
GetSelectionStyle - получить стиль форматирования выделенного текста.
SetSelectionStyle - установить стиль форматирования выделенного текста.
GetSelectionParagraphStyle - получить стиль абзаца выделенного текста.
SetSelectionParagraphStyle - установить стиль абзаца выделенного текста.
SetListBoxSelection - установить выделенную область в объекте списка на новую строчку.
Scroll - выполнить прокрутку текста.
CreateHypertext - создать гиперссылку и действие для неё.
GetFirstHypertext - получить первую гиперссылку в текстовом объекте.
GetNextHypertext - получить следующую гиперссылку в текстовом объекте.
GetText - получить строковое значение текста гиперссылки.
GetNumberHypertext - получить общее количество гиперссылок в текстовом объекте.
RemoveAllHypertext - удалить все гиперссылки в текстовом объекте.
RestoreOriginalText - вернуть содержимое текстового объекта в первоначальное состояние.
PlayAutonarrate - начать автоповествование текста.
StopAutonarrate - завершить автоповествование текста.
IsAutonarratePlaying - проверить, работает ли в данный момент автоповествование.

GetParagraphCount

Получить количество абзацев в текстовом объекте.
Фрагмент текста считается за абзац после каждого перехода на новую строчку (с помощью клавиши **Enter**).

Синтаксис:

GetParagraphCount()

Возврат:

(Ц), количество абзацев в текстовом объекте.

GetLineCount

Получить количество строчек в текстовом объекте.

Синтаксис:

GetLineCount()

Возврат:

(Ц), количество строчек в текстовом объекте.

GetWordCount

Получить количество слов в текстовом объекте.

Синтаксис:

GetWordCount()

Возврат:

(Ц), количество слов в текстовом объекте.

GetTextLength

Получить количество символов в текстовом объекте.

Синтаксис:

GetTextLength()

Возврат:

(Ц), количество символов в текстовом объекте, включая пробелы и переходы на следующую строчку.

ParagraphLength

Получить количество символов в определенном абзаце текстового объекта.

Синтаксис:

ParagraphLength(ParaIndex)

Возврат:

(Ц), количество символов в выбранном абзаце текстового объекта. Если данного абзаца не существует, то **-1** . Переход к следующему абзацу (с помощью клавиши **Enter**) добавляет один символ к общему количеству.

Параметры:

ParaIndex - **(Ц)**, номер абзаца (нумерация начинается с **0**). **НП**, по умолчанию **0**.

LineLength

Получить количество символов в определенной строчке текстового объекта.

Синтаксис:

LineLength(LineIndex)

Возврат:

(Ц), количество символов в выбранной строчке текстового объекта. Если данной строчки не существует, то **-1**. Переход на следующую строчку (с помощью клавиши **Enter**) добавляет один символ к общему количеству.

Параметры:

LineIndex - **(Ц)**, номер строчки (нумерация начинается с **0**). **ОП**.

Общий пример:

```
Debug.trace("Количество абзацев в тексте: "+Text1.GetParagraphCount())
Debug.trace("\n"+"Количество строчек в тексте: "+Text1.GetLineCount())
Debug.trace("\n"+"Количество слов в тексте: "+Text1.GetWordCount())
Debug.trace("\n"+"Количество символов в тексте: "+Text1.GetTextLength())
Debug.trace("\n"+"Количество символов в первом абзаце: "+Text1.ParagraphLength(0))
Debug.trace("\n"+"Количество символов в первой строчке: "+Text1.LineLength(0))
```

ParagraphIndex

Получить номер индексной позиции первого символа в абзаце текстового объекта.

Синтаксис:

ParagraphIndex(ParaIndex)

Возврат:

(Ц), номер индексной позиции первого символа указанного абзаца. Если такого абзаца не существует, то **-1**.

Параметры:

ParaIndex - **(Ц)**, номер абзаца (нумерация начинается с **0**). **НП**, по умолчанию **0**.

LineIndex

Получить номер индексной позиции первого символа в строчке текстового объекта.

Синтаксис:

LineIndex(LineIndex)

Возврат:

(Ц), номер индексной позиции первого символа в указанной строчке.

Параметры:

LineIndex - **(Ц)**, номер строчки (нумерация начинается с **0**). **ОП**.

LineFromChar

Получить номер строчки по номеру индексной позиции символа, расположенного в ней.

Синтаксис:

LineFromChar(CharIndex)

Возврат:

(Ц), номер строчки.

Параметры:

CharIndex - **(Ц)**, номер индексной позиции символа (нумерация начинается с **0**). **ОП**.

PointFromChar

Получить координаты и размер символа по номеру его индексной позиции.

Синтаксис:

PointFromChar(CharIndex)

Возврат:

(О) со следующими свойствами:

x - **(Ч)**, **X**-координата верхнего левого угла символа относительно текст. объекта.

y - **(Ч)**, **Y**-координата верхнего левого угла символа относительно текст. объекта.

width - **(Ч)**, ширина символа.

height - **(Ч)**, высота символа.

Параметры:

CharIndex - **(Ц)**, номер индексной позиции символа (нумерация начинается с **0**). **ОП**.

CharFromPoint

Получить номер индексной позиции символа, ближайшего к указанным координатам.

Синтаксис:

CharFromPoint(PosX, PosY)

Возврат:

(Ц), номер индексной позиции символа, ближайшего к указанным координатам.

Параметры:

PosX - **(Ц)**, **X**-координата верхнего левого угла символа относительно текст. объекта. **ОП**.

PosY - **(Ц)**, **Y**-координата верхнего левого угла символа относительно текст. объекта. **ОП**.

Общий пример:

```
Debug.trace("Номер позиции первого символа во втором абзаце: "+Text1.ParagraphIndex(1))
Debug.trace("\n"+"Номер позиции первого символа во второй строчке: "+Text1.LineIndex(1))
Debug.trace("\n"+"Номер строчки, которая содержит пятый символ: "+Text1.LineFromChar(4))
var C1=Text1.PointFromChar(0) //координаты и размер первого символа текстового объекта
Debug.trace("\n"+"X: "+C1.x+" Y: "+C1.y+"\n"+"Ширина: "+C1.width+"\n"+"Высота: "+C1.height)
Debug.trace("\n"+"Номер символа, ближайшего к (5, 8): "+Text1.CharFromPoint(5, 8))
```

SetSelection и SelectParagraph

Выбрать фрагмент текста в текстовом объекте. Для объекта ввода текста данный фрагмент будет отображаться как выделение. Активной может быть только одна выборка (если сначала выбран фрагмент текста, а затем выбран другой, то выбранным будет текст последнего фрагмента, а с предыдущего фрагмента текста выбор/выделение снимается).

Синтаксис:

SetSelection(Start, End)

SelectParagraph(ParaIndex)

Параметры:

Start - **(Ц)**, номер индексной позиции первого символа во фрагменте (нумерация начинается с **0**). Если **-1**, то выбирается конец текста. **ОП**.

End - **(Ц)**, номер индексной позиции символа следующего за последним во фрагменте. Если **-1**, то выбираются все символы до конца текста. **НП**, по умолчанию равен значению **Start**.

Если **Start** равно **End**, то выбирается не фрагмент, а место для вставки перед символом, чей номер индексной позиции указан в параметрах.

ParaIndex - **(Ц)**, номер абзаца для выбора/выделения (нумерация начинается с **0**). **ОП**.

ReplaceSelection

Заменить, выбранный с помощью функции **SetSelection** или **SelectParagraph**, фрагмент текста в текстовом объекте на указанное строковое значение, либо сделать вставку. Если выбранного фрагмента нет, то происходит вставка строкового значения в начало текста.

Синтаксис:

ReplaceSelection(String)

Параметры:

String - **(С)** для замены/вставки в текстовом объекте. **ОП**.

GetSelectionText

Получить, выбранный с помощью функции **SetSelection** или **SelectParagraph**, фрагмент текста как строковое значение.

Синтаксис:

GetSelectionText()

Возврат:

(С), выбранный фрагмент текста. Если выбранного фрагмента нет, то строковое значение будет пустым.

GetSelection

Получить номера индексных позиций первого и последнего символов фрагмента текста, выбранного с помощью функции **SetSelection** или **SelectParagraph**.

Синтаксис:

GetSelection()

Возврат:

(О) со следующими свойствами:

start - **(Ц)**, номер индексной позиции первого символа во фрагменте. Если выбранного фрагмента нет, то **0**.

end - **(Ц)**, номер индексной позиции символа следующего за последним во фрагменте. Если выбранного фрагмента нет, то **0**.

Общий пример:

```
Text1.SetSelection(-1) //выбор после последнего символа в тексте
Text1.SetSelection(0) //выбор перед первым символом в тексте
Text1.SelectParagraph(0) //выбор первого абзаца в тексте
Text1.SetSelection(1, 1) //выбор между первым и вторым символом в тексте
Text1.SetSelection(0, 1) //выбор первого символа в тексте
Text1.SetSelection(1, 2) //выбор второго символа в тексте
Text1.ReplaceSelection("#")
Debug.trace("Выбранный текст: "+Text1.GetSelectionText())
var txtSel1=Text1.GetSelection()
Debug.trace("\n"+"Номера позиций выбранного фрагмента: "+txtSel1.start+","+txtSel1.end)
```

FindText и FindTextInSelection

Найти символ, слово или фразу в текстовом объекте с указанной позиции. Поиск зависит от регистра символов.

FindText ищет по всему тексту, а **FindTextInSelection** ищет по фрагменту текста, выбранному с помощью функции **SetSelection** или **SelectParagraph**, игнорируя остальную часть (если фрагмент не выбран поиск также будет по всему тексту).

Синтаксис:

```
FindText(String, StartPosition)
FindTextInSelection(String, StartPosition)
```

Возврат:

(Ц), номер индексной позиции первого символа искомого текста, если он найден, иначе **-1**.

Параметры:

String - **(С)** для поиска в тексте. **ОП**.

StartPosition - **(Ц)**, номер индексной позиции символа (нумерация начинается с **0**), с которой начнется поиск. **НП**, по умолчанию **0**.

Пример:

```
var Word1="Tiger", WordCount=0
var FindResult=Text1.FindText(Word1, 0)
if (FindResult==(-1)) {WordCount=0}
else {while (FindResult!=(-1))
{WordCount=WordCount+1
FindResult=Text1.FindText(Word1, FindResult+1)}}
Debug.trace("Найдено слов: "+WordCount)
```

SetColour

Изменить цвет текстового объекта.

Синтаксис:

```
SetColour(Colour)
```

Параметры:

Colour - цвет текста. **НП**, по умолчанию **0** (чёрный).

Варианты значения параметра:

1) **(С)**, один из восьми цветов (**"white"**, **"black"**, **"red"**, **"green"**, **"blue"** , **"yellow"**, **"cyan"**, **"magenta"**).
2) **(С)**, шестнадцатеричный код цвета **HTML**, обязательно со знаком [#] (например коричневый: **"#A52A2A"**).
3) **(Ц)**, значение функции **RGB** (функцию можно вставить в параметр).
4) три **(Ц)** через запятую, для **RGB** кодировки цвета (например серый: **128, 128, 128**).
5) специальное значение **-1**, убирающее текущий цвет.

Пример:

```
wait(1);Text1.SetColour("red")
wait(1);Text1.SetColour("#F27600")
wait(1);Color1=RGB(253, 250, 0);Text1.SetColour(Color1)
wait(1);Text1.SetColour(0, 170, 65)
wait(1);Text1.SetColour(-1)
```

GetSelectionStyle

Получить в текстовом объекте стиль фрагмента текста, выбранного с помощью функции **SetSelection** или **SelectParagraph**. Данный стиль можно применять к фрагментам других текстовых объектов с помощью функции **SetSelectionStyle**.

Синтаксис:

GetSelectionStyle()

Возврат:

(О) с некоторыми из следующих свойств:

bold - **(Б)**, жирный текст (**true** - да, **false** - нет).

italic - **(Б)**, текст курсивом (**true** - да, **false** - нет).

underline - **(Б)**, текст с подчеркиванием (**true** - да, **false** - нет).

subscript - **(Б)**, текст в нижнем индексе (**true** - да (пример: цифра **2** в хим. формуле воды), **false** - нет).

superscript - **(Б)**, текст в верхнем индексе (**true** - да (пример: показатель степени в математике), **false** - нет).

fontname - **(С)**, название шрифта.

fontsize - **(Ц)**, размер шрифта в пунктах (**pt**).

fontaspect - **(Ц)**, аспект шрифта.

colour - **(С)**, цвет текста.

backgroundcolour - **(С)**, цвет фона текста или **"transparent"** для текста без фона.

shadowcolour - **(С)**, цвет тени текста или **"none"** для текста без тени.

Цвета задаются только шестнадцатеричным кодом цвета **HTML**, без знака [#] (например коричневый: **"A52A2A"**). **Использование числовых значений RGB возможно, но приведет к неправильному цвету.**

ВАЖНО:

1) Для получения свойств, необходимо, чтобы фрагмент текста в текстовом объекте был выбран функцией **SetSelection**, иначе значения всех свойств будут **undefined** (неопределенными).

2) Для получения значения свойства необходимо, чтобы во всем выбранном фрагменте форматирование было одинаково по данному свойству. Например, если часть текста в выбранном фрагменте выделена жирным, а часть нет, то значение свойства выделения жирным у данного фрагмента будет **undefined**.

SetSelectionStyle

Установить в текстовом объекте стиль для фрагмента текста, выбранного с помощью функции **SetSelection** или **SelectParagraph**. Возможно изменение не всего стиля, а конкретных свойств.

Синтаксис:

SetSelectionStyle(Style)

Параметры:

Style - **(О)** со всеми или несколькими свойствами (описаны у функции **GetSelectionStyle**) для применения. **ОП**.

Общий пример:

```
Text1.SetSelection(0, -1) //выбрать весь 1-ый текст целиком
var txtStyle1=Text1.GetSelectionStyle()
var txtStyle2=new Object()
wait(3);txtStyle2.colour="FFFF00"
Text1.SetSelectionStyle(txtStyle2) //1-му тексту назначить новое свойство стиля: желтый цвет
wait(3);Text2.SetSelection(0, -1) //выбрать весь 2-ой текст целиком
Text2.SetSelectionStyle(txtStyle1) //2-му тексту присвоить все старые свойства стиля 1-го текста
//Вывести значения всех свойств стиля 1-го текста
Debug.trace("Жирность: "+txtStyle1.bold+"\n")
Debug.trace("Курсив: "+txtStyle1.italic+"\n")
Debug.trace("Подчеркивание: "+txtStyle1.underline+"\n")
Debug.trace("Нижний индекс: "+txtStyle1.subscript+"\n")
Debug.trace("Верхний индекс: "+txtStyle1.superscript+"\n")
Debug.trace("Название шрифта: "+txtStyle1.fontname+"\n")
Debug.trace("Размер шрифта: "+txtStyle1.fontsize+"\n")
Debug.trace("Аспект шрифта: "+txtStyle1.fontaspect+"\n")
Debug.trace("Цвет текста: "+txtStyle1.colour+"\n")
Debug.trace("Цвет фона текста: "+txtStyle1.backgroundcolour+"\n")
Debug.trace("Цвет тени текста: "+txtStyle1.shadowcolour)
```

GetSelectionParagraphStyle

Получить в текстовом объекте стиль абзаца во фрагменте текста, выбранном с помощью функции **SetSelection** или **SelectParagraph**. Данный стиль абзаца можно применять к фрагментам других текстовых объектов с помощью функции **SetSelectionParagraphStyle**.

Синтаксис:

GetSelectionParagraphStyle()

Возврат:

(O) с некоторыми из следующих свойств:

justification - **(C)**, выравнивание текста, возможные значения:
"Left" (по левому краю), **"Right"** (по правому краю), **"Centre"** (по центру),
"Full" (по ширине), **"Density"** (разуплотнённое) или **"Aspect"** (растянутое).

linespacing - **(C)**, междустрочный интервал, возможные значения:
"xN", где **N** - кратность (например **"x1"** или **"x1.5"**);
"=Npt", где **N** - точная высота в пунктах (например **"=12pt"** или **"=14pt"**);
"At Least" - определённый интервал, зависящий от размера шрифта.

spacingbefore - **(Ц)**, промежуток (в пунктах) между текущим абзацем и предыдущим.

spacingafter - **(Ц)**, промежуток (в пунктах) между текущим абзацем и следующим.

leftindent - **(Ц)**, промежуток (в пикселях) от левого края рамки текстового объекта до текста абзаца.

rightindent - **(Ц)**, промежуток (в пикселях) от правого края рамки текстового объекта до текста абзаца.

firstlineindent - **(Ц)**, отступ первой строчки абзаца (в пикселях).

hangingindent - **(Ц)**, выступ абзаца (в пикселях).

bullet.type - **(C)**, тип маркера абзаца, возможные значения:
"NoBullet" (без маркера), **"Character"** (символ как маркер) или **"Graphical"** (изображение как маркер).

bullet.character - **(C)**, символ, использующийся для маркера абзаца.

bullet.font - **(C)**, шрифт символа, использующегося для маркера абзаца.

bullet.colour - **(C)**, цвет маркера абзаца, как шестнадцатеричный код цвета **HTML**, без знака [#] (например коричневый: **"A52A2A"**).

ВАЖНО:

1) Для получения свойств, необходимо, чтобы фрагмент текста в текстовом объекте был выбран функцией **SetSelection**, иначе значения всех свойств будут **undefined** (неопределенными).

2) Для получения значения свойства необходимо, чтобы во всем выбранном фрагменте форматирование было одинаково по данному свойству. Например, если у одного абзаца в выбранном фрагменте выравнивание по ширине, а у другого по центру, то значение свойства выравнивания у данного фрагмента будет **undefined**.

Пример:

```
//Алгоритм выбора нужного абзаца целиком
var parNum=0
var parBegin=Text1.ParagraphIndex(parNum)
var parEnd=parBegin+Text1.ParagraphLength(parNum)-1
Text1.SetSelection(parBegin, parEnd)
//Получить и отобразить значения всех свойств выбранного абзаца
var parStyle=Text1.GetSelectionParagraphStyle()
Debug.trace("Выравнивание: "+parStyle.justification+"\n")
Debug.trace("Междустрочный интервал: "+parStyle.linespacing+"\n")
Debug.trace("Разрыв между текущим абзацем и предыдущим: "+parStyle.spacingbefore+"\n")
Debug.trace("Разрыв между текущим абзацем и следующим: "+parStyle.spacingafter+"\n")
Debug.trace("От левого края до абзаца: "+parStyle.leftindent+"\n")
Debug.trace("От правого края до абзаца: "+parStyle.rightindent+"\n")
Debug.trace("Отступ первой строчки абзаца: "+parStyle.firstlineindent+"\n")
Debug.trace("Выступ абзаца: "+parStyle.hangingindent+"\n")
Debug.trace("Тип маркера: "+parStyle.bullet.type+"\n")
Debug.trace("Символ маркера: "+parStyle.bullet.character+"\n")
Debug.trace("Шрифт маркера: "+parStyle.bullet.font+"\n")
Debug.trace("Цвет маркера: "+parStyle.bullet.colour)
```

SetSelectionParagraphStyle

Установить в текстовом объекте стиль для абзацев, чьи части входят во фрагмент текста, выбранный с помощью функции **SetSelection** или **SelectParagraph**. Для назначения стиля всему абзацу достаточно, чтобы был выбран хотя бы один символ из него. Возможно изменение не всего стиля, а конкретных свойств.

ВАЖНО:

1) Изменение отдельно свойства междустрочного интервала в абзаце **работает только с кратными значениями**.

2) Изменение отдельно любых свойств маркера абзаца **не работает вообще и приводит к ошибке**.

3) При передаче всех свойств от одного текстового объекта к другому, ошибок не происходит, этим можно воспользоваться для переназначения свойств маркера абзаца.

Синтаксис:

SetSelectionParagraphStyle(Style)

Параметры:

Style - **(О)** со всеми или несколькими свойствами (описаны у функции **GetSelectionParagraphStyle**) для применения. **ОП**.

Пример:

```
//Алгоритм выбора нужного абзаца целиком
var parNum=0;var parBegin=Text1.ParagraphIndex(parNum)
var parEnd=parBegin+Text1.ParagraphLength(parNum)-1
Text1.SetSelection(parBegin, parEnd)
//Изменение свойств стиля абзаца
var parStyle1=Text1.GetSelectionParagraphStyle()
var parStyle2=new Object();parStyle2.justification="Centre"
wait(3);Text1.SetSelectionParagraphStyle(parStyle2) //абзацу 1-го текста назначить свойство стиля
wait(3);Text2.SetSelection(0, -1) //выбрать весь 2-ой текст
Text2.SetSelectionParagraphStyle(parStyle1) //присвоить 2-му тексту свойства стиля 1-го текста
```

SetListBoxSelection

Выделить конкретную строчку в указанном текстовом объекте списка (**List Box**) или снять выделение. Использование функции с другими текстовыми объектами не даст эффекта.

Синтаксис:

SetListBoxSelection(Index)

Параметры:

Index - **(Ц)**, номер строчки списка для выделения (нумерация начинается с **0**) или **-1** для снятия выделения. **ОП**.

Пример:

```
Listbox1.SetListBoxSelection(1)
```

Scroll

Прокрутить текст в текстовом объекте.

Синтаксис:

Scroll(Command, Amount)

Параметры:

Command - **(С)**, команда прокрутки, возможные значения: **"LineUp"**, **"LineDown"**, **"LineTo"**, **"ParagraphUp"**, **"ParagraphDown"**, **"PageUp"**, **"PageDown"**, **"Start"**, **"End"**. **ОП**.

Amount - **(Ц)**, показатель прокрутки.

Для команд **"LineUp"** и **"LineDown"** - число строчек для прокрутки. **НП**, по умолчанию **1**.

Для команды **"LineTo"** - номер строчки (нумерация начинается с **0**), куда следует совершить прокрутку. **НП**, по умолчанию **0**.

Не влияет на команды **"Start"**, **"End"** (прокрутка только в начало/конец) и **"ParagraphUp"**, **"ParagraphDown"**, **"PageUp"**, **"PageDown"** (прокрутка всегда только на **один** абзац/страницу).

Пример:

```
wait(3);Text1.Scroll("ParagraphDown") //прокрутить текст на один абзац вниз
wait(3);Text1.Scroll("LineTo", 3) //прокрутить текст к четвертой строчке
wait(3);Text1.Scroll("End") //прокрутить текст до самого конца
```

CreateHypertext

Создать в текстовом объекте гиперссылку, во фрагменте текста, выбранном с помощью функции **SetSelection** или **SelectParagraph**, и назначить действие, которое будет выполняться при нажатии левой кнопкой мыши на эту гиперссылку.

Синтаксис:

CreateHypertext(Action)

Параметры:

Action - **(С)**, любой фрагмент кода, который выполнится при нажатии левой кнопкой мыши на гиперссылку. **ОП**.

ВАЖНО: Если должна выполниться одна единственная пользовательская функция **не имеющая параметров**, то можно указать **только её название** без скобок, а также без кавычек (не как строковое значение). Если же пользовательская функция **имеет параметры**, то её следует указать как **фрагмент кода** в виде строкового значения.

GetFirstHypertext

Получить первую (самую ближнюю к концу текста) гиперссылку в текстовом объекте.

Синтаксис:

GetFirstHypertext()

Возврат:

(О), первая гиперссылка или **null**, если в текстовом объекте гиперссылок нет.

GetNextHypertext

Получить следующую гиперссылку в текстовом объекте. Для работы необходимо, чтобы была получена предыдущая гиперссылка с помощью функции **GetFirstHypertext** или **GetNextHypertext**. Порядок расположения гиперссылок идет от конца к началу (первой считается самая ближняя к концу текста, последней - самая ближняя к началу).

Синтаксис:

GetNextHypertext(Link)

Возврат:

(О), следующая гиперссылка или **null**, если далее в текстовом объекте гиперссылок больше нет.

Параметры:

Link - **(О)**, предыдущая гиперссылка.

GetText

Получить текст, содержащийся в гиперссылке, как строковое значение.

Синтаксис:

GetText()

Возврат:

(С), текст, содержащийся в гиперссылке.

GetNumberHypertext

Получить количество гиперссылок в текстовом объекте.

Синтаксис:

GetNumberHypertext()

Возврат:

(Ц), количество гиперссылок в текстовом объекте.

Общий пример:

```
//Пользовательские функции задаются только в скриптовом объекте страницы
function Action1() {Text1.SetColour("yellow")} //создать пользовательскую функцию без параметров
Text1.SetSelection(2, 8); Text1.CreateHypertext(Action1) //гиперссылка с функцией
Text1.SetSelection(18, 24); Text1.CreateHypertext('Text1.SetColour("red")') //гиперссылка с кодом
var Link1=Text1.GetFirstHypertext(); var Link2=Text1.GetNextHypertext(Link1)
Debug.trace("Текст первой гиперссылки: "+Link1.GetText()+"\n")
Debug.trace("Текст второй гиперссылки: "+Link2.GetText()+"\n")
Debug.trace("Количество гиперссылок: "+Text1.GetNumberHypertext())
```

RemoveAllHypertext

Убрать все гиперссылки в текстовом объекте.

Синтаксис:

RemoveAllHypertext()

Пример:

```
Text1.RemoveAllHypertext()
```

RestoreOriginalText

Вернуть содержимое текстового объекта к исходному состоянию.

Синтаксис:

RestoreOriginalText()

Пример:

```
Text1.RestoreOriginalText()
```

PlayAutonarrate

Начать воспроизведение автоповествования текстового объекта.

Синтаксис:

PlayAutonarrate()

Пример:

```
Text1.PlayAutonarrate()
```

StopAutonarrate

Остановить воспроизведение автоповествования текстового объекта.

Синтаксис:

StopAutonarrate()

Пример:

```
Text1.StopAutonarrate
```

IsAutonarratePlaying

Проверить, воспроизводится ли сейчас автоповествование текстового объекта.

Синтаксис:

IsAutonarratePlaying()

Возврат:

(Б), состояние воспроизведения автоповествования (**true** - воспроизводится, **false** - нет).

Пример:

```
Debug.trace("Проверка работы автоповествования: "+Text1.IsAutonarratePlaying())
```

Кнопки

Кнопки настраиваются на вкладке **"Button"** диалогового окна **"Properties"**.

Типы кнопок:

Нажимная (Push Button) - "нажатое" состояние только если на неё наведен курсор и нажата левая кнопка мыши, иначе состояние "ненажатое".

Переключатель (Check Box) - кнопка системы включить/выключить.

Радио-кнопки (Radio Button) - группа кнопок, из которых только одна может иметь "нажатое" состояние на текущий момент.

RADIO_GROUP_#_ID - **(Ц)**, номер (нумерация начинается с **0**) текущей нажатой радио-кнопки в группе под номером **#** (нумерация начинается с **1**), **-1** если ни одна не нажата.

RADIO_GROUP_#_NAME - **(С)**, название текущей нажатой радио-кнопки в группе под номером **#** (нумерация начинается с **1**), **null** если ни одна не нажата.

ВАЖНО: в свойствах есть возможность выбрать группу под номером **0**, но для неё нет системных переменных.

Список функций:

GetState - получить текущее состояние "нажатости" кнопки.

SetState - установить состояние "нажатости" кнопки.

GetTextObject - получить текстовый объект, находящийся внутри кнопки.

GetState

Получить текущее состояние **"нажатости"** кнопки.

Синтаксис:

```
GetState()
```

Возврат:

(Б), состояние объекта кнопки (**true** - нажатое, **false** - ненажатое).

Пример:

```
Debug.trace("Состояние кнопки: "+Button1.GetState())
while (Button1.GetState()==false) {wait(0.01)}
Debug.trace("\n"+"Состояние кнопки: "+Button1.GetState())
```

SetState

Установить состояние **"нажатости"** кнопки.

Синтаксис:

```
SetState(State)
```

Параметры:

State - **(Б)**, состояние объекта кнопки (**true** - нажатое, **false** - ненажатое). **ОП**.

Пример:

```
wait(3);Button1.SetState(true) //кнопка нажмётся сама через 3 секунды
```

GetTextObject

Получить текстовый объект, находящийся внутри кнопки для дальнейшего применения к нему функций. При взаимодействии кнопки с курсором мыши, текстовый объект возвращается в исходное состояние.

Синтаксис:

```
GetTextObject()
```

Возврат:

(О), текстовый объект внутри объекта кнопки.

Пример:

```
while (true) {wait(1);Button1.GetTextObject().SetColour(RGB(0, 170, 65))}
```

Слайд-шоу

Слайд-шоу является набором различных изображений, которые можно переключать.
Поддерживаемые форматы:
BMP, **CGM**, **JPG**, **JPEG**, **PCX**, **PNG**, **TGA**, **GIF**, **TIF**, **TIFF**, **PCD**, **WMF**, **EMF**, **ILV**.

Список функций:
Play - воспроизвести слайд-шоу.
Stop - остановить слайд-шоу.
Pause - поставить на паузу слайд-шоу.
IsPlaying - проверить воспроизводится ли слайд-шоу.
Continue - продолжить воспроизведение слайд-шоу.
GetSlide - получить слайд, видимый в данный момент.
GetSlideCount - получить количество слайдов в слайд-шоу.
GotoSlide - перейти на указанный слайд.
UpdateFiles - обновить файлы слайд-шоу.

Play

Начать воспроизведение слайд-шоу с определенного слайда.
Синтаксис:
Play(Slide)
Параметры:
Slide - **(Ц)**, номер слайда (нумерация начинается с **0**), с которого начнётся воспроизведение. **НП**, по умолчанию **0**.
Пример:
```
Slideshow1.Play()
```

Stop

Остановить воспроизведение слайд-шоу.
Слайд-шоу останавливается на текущем слайде и не сбрасывается на первый слайд в слайд-шоу.
Синтаксис:
Stop()
Пример:
```
Slideshow1.Stop()
```

Pause

Приостановить воспроизведение слайд-шоу.
Синтаксис:
Pause()
Пример:
```
Slideshow1.Pause()
```

Continue

Продолжить воспроизведение слайд-шоу со слайда, на котором была выполнена остановка или приостановка.
Синтаксис:
Continue()
Пример:
```
Slideshow1.Continue()
```

IsPlaying

Проверить, воспроизводится ли указанное слайд-шоу в данный момент или нет.
Синтаксис:
IsPlaying()
Возврат:
(Б), состояние воспроизведения слайд-шоу (**true** - воспроизводится, **false** - нет).
Пример:
```
Debug.trace("Воспроизводится ли слайд-шоу? "+Slideshow1.IsPlaying())
```

GotoSlide

Перейти к определенному слайду в слайд-шоу.

Синтаксис:

GotoSlide(Slide)

Параметры:

Slide - **(Ц)**, номер слайда для перехода (нумерация начинается с **0**), **-1** для последнего слайда. **ОП**.

Пример:

```
Slideshow1.GotoSlide(3) //перейти на четвертый слайд
```

GetSlide

Получить порядковый номер текущего слайда в слайд-шоу.

Синтаксис:

GetSlide()

Возврат:

(Ц), порядковый номер текущего слайда (нумерация начинается с **0**).

Пример:

```
Debug.trace("Текущий слайд: "+Slideshow1.GetSlide()
```

GetSlideCount

Получить количество слайдов в слайд-шоу.

Синтаксис:

GetSlideCount()

Возврат:

(Ц), количество слайдов в слайдшоу.

Пример:

```
Debug.trace("Количество слайдов: "+Slideshow1.GetSlideCount())
```

UpdateFiles

Обновить файлы в слайд-шоу (только если в свойствах объекта **"Properties"** на вкладке **"Slideshow"** стоит флажок **"All Files"**). Уже воспроизводимое слайд-шоу остановится со сбросом к первому слайду.

Синтаксис:

UpdateFiles()

Пример:

```
Slideshow1.UpdateFiles()
```

Кадры и векторные объекты

Специальные функции объекта кадр предназначены для рисования векторных полигонов внутри кадра.

Особенности:

1) Полигоны, рисуемые скриптом, располагаются относительно координат пространства страницы, а не кадра. В кадр попадут только те части полигона, которые соответствуют пересечению пространства кадра и той области страницы, где должны располагаться полигоны.
2) Изначально созданный, искаженный (с поворотом, сдвигом и прочим) кадр повлияет на отрисовку полигонов (они тоже исказятся), если он соответсвует по положению той области страницы, где рисуются полигоны.
3) При перемещении кадра уже после отрисовки полигонов, переместятся все полигоны попавшие в кадр. Если нарисовать полигоны по координатам за пределами кадра, а затем переместить кадр в то место, где они нарисованы, то эти полигоны в кадре не появятся.
4) На пересечении в одном кадре двух полигонов с заливкой образуется пустота.
5) В одном кадре все полигоны всегда одного цвета и стиля. Для рисования нескольких независимых полигонов, следует использовать несколько объектов кадра.
5) Функции векторного рисования предназначены только для объекта кадр и не действуют с векторными объектами.
6) Чем больше полигонов нарисовано в кадре, тем медленнее будут рисоваться новые.

Список функций:

DrawLine - нарисовать линию.
DrawRectangle - нарисовать прямоугольник.
DrawRoundRectangle - нарисовать прямоугольник со скругленными углами.
DrawEllipse - нарисовать эллипс или дугу.
DrawStar - нарисовать звезду.
DrawRegPoly - нарисовать многоугольник.
AddPoint - добавить точку к полигону.
MovePoint - переместить точку в полигоне.
RemovePoint - убрать точку в полигоне.
ClearDraw - очистить кадр от всех нарисованных скриптовых полигонов.
SetLineStyle - назначить стиль линий для всех скриптовых полигонов в кадре.
SetLineColour - назначить цвет линий для всех скриптовых полигонов в кадре.
SetFillColour - назначить цвет заливки для всех скриптовых полигонов в кадре.

Векторные изображения можно рисовать как в самой **программе**, так и импортировать, созданные в других векторных редакторах (лучшая совместимость с векторными изображениями в **WMF**-формате, созданными в **Serif DrawPlus X8**). Каждый векторный объект состоит из множества дочерних элементов - полигонов. Изменять с помощью скрипта можно только два свойства полигонов: цвет заливки и цвет линии.

Список функций:

SetLineColour - назначить цвет линии полигона в векторном объекте.
SetFillColour - назначить цвет заливки полигона в векторном объекте.

DrawLine

Нарисовать линию.

Синтаксис:

DrawLine(StartX, StartY, EndX, EndY)
DrawLine(Start, End)

Параметры:

StartX - **(Ц)**, **X**-координата начала линии. **ОП**.
StartY - **(Ц)**, **Y**-координата начала линии. **ОП**.
EndX - **(Ц)**, **X**-координата конца линии. **ОП**.
EndY - **(Ц)**, **Y**-координата конца линии. **ОП**.
Start - **(О)** со свойствами **x** и **y** - **(Ц)**, координаты начала линии. **ОП**.
End - **(О)** со свойствами **x** и **y** - **(Ц)**, координаты конца линии. **ОП**.

Пример:

```
var Pos=Frame1.GetPosition()
var Start=new Object(); var End=new Object()
Start.x=Pos.x-20; Start.y=Pos.y-20
End.x=Pos.x+20; End.y=Pos.y+20
Frame1.DrawLine(Start, End)
Frame1.DrawLine(Start.x, Start.y+20, End.x, End.y+20)
```

DrawRectangle

Нарисовать прямоугольник.

Синтаксис:

DrawRectangle(PosX, PosY, Width, Height)

Параметры:

PosX - **(Ц)**, **X**-координата верхнего левого угла прямоугольника. **ОП**.
PosY - **(Ц)**, **Y**-координата верхнего левого угла прямоугольника. **ОП**.
Width - **(Ц)**, ширина прямоугольника. **ОП**.
Height - **(Ц)**, высота прямоугольника. **ОП**.

Пример:

```
var Pos=Frame1.GetPosition()
Frame1.DrawRectangle(Pos.x-20, Pos.y-20, 20, 40)
```

DrawRoundRectangle

Нарисовать прямоугольник со скругленными углами.

Синтаксис:

DrawRoundRectangle(PosX, PosY, Width, Height, Radius)

Параметры:

PosX - **(Ц)**, **X**-координата верхнего левого угла прямоугольника. **ОП**.
PosY - **(Ц)**, **Y**-координата верхнего левого угла прямоугольника. **ОП**.
Width - **(Ц)**, ширина прямоугольника. **ОП**.
Height - **(Ц)**, высота прямоугольника. **ОП**.
Radius - **(Ц)**, радиус углов. **ОП**.

Пример:

```
var Pos=Frame1.GetPosition()
Frame1.DrawRoundRectangle(Pos.x-20, Pos.y-20, 40, 20, 5)
```

При рисовании эллипса, звезды или многоугольника учитываются не координаты их центра, а координаты верхнего левого угла ограничительного прямоугольника, в который фигура вписана. Многоугольники/звезды с нечётным числом сторон/лучей рисуются **со смещением** вверх от ограничительного прямоугольника (особенно выражено у треугольников, пятиугольников и трёхконечных, пятиконечных звезд). Чтобы **убрать смещение**, к параметру **PosY** следует прибавить значение:
Hight*0.165 - для треугольников и трехконечных звезд;
Hight*0.05 - для пятиугольников и пятиконечных звезд.

DrawEllipse

Нарисовать эллипс или дугу.
Синтаксис:
DrawEllipse(PosX, PosY, Width, Height, Shape)
Параметры:
PosX - **(Ц)**, **X**-координата, верхнего левого угла ограничительного прямоугольника. **ОП**.
PosY - **(Ц)**, **Y**-координата, верхнего левого угла ограничительного прямоугольника. **ОП**.
Width - **(Ц)**, ширина ограничительного прямоугольника. **ОП**.
Height - **(Ц)**, высота ограничительного прямоугольника. **ОП**.
Shape - **(Ц)**, тип кривой для рисования. **НП**, по умолчанию **0**.
Возможные значения:
0 - закрытый эллипс.
1 - горизонтальная дуга.
2 - вертикальная дуга.
3 - синусоидальная волна.
Пример:

```
var Pos=Frame1.GetPosition()
Frame1.DrawEllipse(Pos.x-20, Pos.y-20, 20, 40, 0) //эллипс
Frame1.DrawEllipse(Pos.x-20, Pos.y-20, 40, 20, 3) //волна
```

DrawStar

Нарисовать звезду.
Синтаксис:
DrawStar(PosX, PosY, Width, Height, Points)
Параметры:
PosX - **(Ц)**, **X**-координата, верхнего левого угла ограничительного прямоугольника. **ОП**.
PosY - **(Ц)**, **Y**-координата, верхнего левого угла ограничительного прямоугольника. **ОП**.
Width - **(Ц)**, ширина ограничительного прямоугольника. **ОП**.
Height - **(Ц)**, высота ограничительного прямоугольника. **ОП**.
Points - **(Ц)**, количество лучей звезды (от **2** до **100**). **ОП**.
Пример:

```
var Pos=Frame1.GetPosition()
Frame1.DrawStar(Pos.x-10, Pos.y-20, 20, 40, 2) //ромб
Frame1.DrawStar(Pos.x-20, Pos.y-20, 40, 40, 8) //восьмиконечная звезда
```

DrawRegPoly

Нарисовать многоугольник.
Синтаксис:
DrawRegPoly(PosX, PosY, Width, Height, Sides)
Параметры:
PosX - **(Ц)**, **X**-координата, верхнего левого угла ограничительного прямоугольника. **ОП**.
PosY - **(Ц)**, **Y**-координата, верхнего левого угла ограничительного прямоугольника. **ОП**.
Width - **(Ц)**, ширина ограничительного прямоугольника. **ОП**.
Height - **(Ц)**, высота ограничительного прямоугольника. **ОП**.
Sides - **(Ц)**, количество сторон многоугольника. (от **3** до **50**). **ОП**.
Пример:

```
var Pos=Frame1.GetPosition()
Frame1.DrawRegPoly(Pos.x-20, Pos.y-20+40*0.165, 40, 40, 3) //треугольник
Frame1.DrawRegPoly(Pos.x-20, Pos.y-20, 40, 40, 6) //шестиугольник
```

AddPoint

Добавить новую точку в полигон, нарисованный с помощью скрипта.

Синтаксис:

AddPoint(Index, PosX, PosY)
AddPoint(Index, Pos)

Параметры:

Index - **(Ц)**, порядковый номер существующей точки для добавления новой после неё (нумерация начинается с **0**).
Вместо номера последней точки можно использовать **-1**. **ОП**.
PosX - **(Ц)**, **X**-координата новой точки. **ОП**.
PosY - **(Ц)**, **Y**-координата новой точки. **ОП**.
Pos - **(О)** со свойствами **x** и **y** - **(Ц)**, координаты новой точки. **ОП**.

Пример:

```
wait(3); Frame1.AddPoint(-1, Frame1.GetPosition())
```

MovePoint

Переместить существующую точку полигона, нарисованного с помощью скрипта.

Синтаксис:

MovePoint(Index, PosX, PosY)
MovePoint(Index, Pos)

Параметры:

Index - **(Ц)**, порядковый номер перемещаемой существующей точки (нумерация начинается с **0**).
Вместо номера последней точки можно использовать **-1**. **ОП**.
PosX - **(Ц)**, новая **X**-координата перемещаемой точки. **ОП**.
PosY - **(Ц)**, новая **Y**-координата перемещаемой точки. **ОП**.
Pos - **(О)** со свойствами **x** и **y** - **(Ц)**, координаты перемещаемой точки. **ОП**.

Пример:

```
wait(3); Frame1.MovePoint(-1, Frame1.GetPosition())
```

RemovePoint

Удалить существующую точку из полигона, нарисованного с помощью скрипта.

Синтаксис:

RemovePoint(Index)

Параметры:

Index - **(Ц)**, порядковый номер удаляемой существующей точки (нумерация начинается с **0**).
Вместо номера последней точки можно использовать **-1**. **НП**, по умолчанию **-1**. Точку номер **0** удалить нельзя.

Пример:

```
Frame1.RemovePoint()
```

ClearDraw

Очистить кадр от всех полигонов, нарисованных с помощью скрипта.

Синтаксис:

ClearDraw()

Пример:

```
Frame1.ClearDraw()
```

SetLineStyle

Установить ширину и тип линий для всех полигонов кадра, нарисованных с помощью скрипта.

Синтаксис:

SetLineStyle(Width, Style, Dash, Shift)

Параметры:

Width - **(Ц)**, ширина линии. **ОП**.

Style - **(Ц)**, тип линии (**0** - сплошная, **1** - пунктирная, **2** - штриховая). **НП**, по умолчанию **0**.

Dash - **(Ц)**, длина в пикселях: пробела - для пунктирной линии, штриха и пробела - для штриховой линии. **НП**, по умолчанию **0**.

Shift - **(Ц)**, сдвиг пунктирной линии. **НП**, по умолчанию **0**.

Пример:

```
Frame1.SetLineStyle(4, 2, 16)
```

SetLineColour и SetFillColour

Изменить цвет линии/заливки полигона.

1) При использовании функции с полигонами, нарисованными скриптом в кадре, цвет линии/заливки применяется на все полигоны кадра, поэтому, чтобы сделать полноценное разноцветное изображение, следует создать несколько кадров.

2) При использовании функции с полигонами векторного объекта, изменяется цвет линии/заливки только указанного полигона.

Синтаксис:

Изменение цвета линии

```
SetLineColour(Colour, Transparency)
```

Изменение цвета заливки

```
SetFillColour(Colour, Transparency)
```

Параметры:

Colour - цвет линии/заливки. **ОП**.

Варианты значения параметра:

1) **(С)**, один из восьми цветов (**"white"**, **"black"**, **"red"**, **"green"**, **"blue"**, **"yellow"**, **"cyan"**, **"magenta"**).

2) **(С)**, шестнадцатеричный код цвета **HTML** (например коричневый: **"#A52A2A"**).

3) **(Ц)**, значение функции **RGB** (функцию можно вставить в параметр).

4) три **(Ц)** через запятую, для **RGB** кодировки цвета (например серый: **128, 128, 128**).

5) специальное значение **-1**, убирающее текущую линию/заливку (у скриптовых полигонов сбрасывается также стиль линий на значение по умолчанию, а дальнейшее рисование без смены цвета вызывает ошибку).

Transparency - **(Ц)** от **0** до **100** (**0** - полная непрозрачность, **100** - полная прозрачность, а промежуточные значения - различная степень прозрачности), прозрачность линии/заливки. **НП**, по умолчанию **0**.

Примеры:

```
//1) Изменить заливку всех полигонов кадра
wait(2);Frame1.SetFillColour("red")
wait(2);Frame1.SetFillColour("#EF5C00")
wait(2);Frame1.SetFillColour(RGB(255, 255, 0))
wait(2);var Color1=RGB(0, 255, 0)
Frame1.SetFillColour(Color1)
wait(2);Frame1.SetFillColour(0, 192, 192)
var r=0, g=0, b=255
wait(2);Frame1.SetFillColour(r, g, b)
wait(2);Frame1.SetFillColour(-1)

//2) Изменить цвет и заливку первого полигона векторного объекта
Vector1.GetFirstChild().SetLineColour("blue")
Vector1.GetFirstChild().SetFillColour("yellow")
```

Мультикадры

Мультикадры - графические объекты, содержащие внутри себя несколько объектов кадра, между которыми можно переключаться. События, происходящие в отдельных кадрах не сбрасываются при переключении кадров.

Список функций:

Play - начать проигрывание кадров друг за другом.
Stop - остановить проигрывание кадров.
Forward - переключиться на следующий кадр.
Backward - переключиться на предыдущий кадр.
ToStart - переключиться на начальный кадр.
ToEnd - переключиться на последний кадр.
ToFrame - переключиться на указанный кадр.
ToRandom - переключиться на случайный кадр.

Play

Начать проигрывание мультикадра (переключение кадров друг за другом). Скорость и количество повторов проигрывания задаются в свойствах объекта **"Properties"** на вкладке **"Multiframe"**.

Синтаксис:

Play()

Пример:

```
MultiFrame1.Play()
```

Stop

Остановить на текущем кадре проигрывание (переключение кадров) мультикадра.

Синтаксис:

Stop()

Пример:

```
MultiFrame1.Stop()
```

Forward

Переключиться на следующий кадр мультикадра. Если текущий кадр последний, то произойдет переход к начальному кадру.

Синтаксис:

Forward()

Пример:

```
MultiFrame1.Forward()
```

Backward

Переключиться на предыдущий кадр мультикадра. Если текущий кадр начальный, то произойдет переход к последнему кадру.

Синтаксис:

Backward()

Пример:

```
MultiFrame1.Backward()
```

ToStart

Переключиться на начальный кадр мультикадра.

Синтаксис:

ToStart()

Пример:

MultiFrame1.ToStart()

ToEnd

Переключиться на последний кадр мультикадра.

Синтаксис:

ToEnd()

Пример:

MultiFrame1.ToEnd()

ToFrame

Переключиться на определенный кадр мультикадра.

Синтаксис:

ToFrame(Number)

Параметры:

Number - **(Ц)**, номер кадра в мультикадре (нумерация начинается с **0**). **ОП**.

Пример:

MultiFrame1.ToFrame(1) //переключиться на 2-ой кадр

ToRandom

Переключиться на случайный кадр мультикадра (без повторений, пока не будут показаны все кадры, затем новая случайная последовательность без повторений).

Синтаксис:

ToRandom()

Пример:

MultiFrame1.ToRandom()

Tween-анимация

Графический объект **Tween** - специальный кадр предназначенный для анимации. Может включать в себя любые графические объекты, в том числе другие объекты анимации. Временная шкала объекта анимации не имеет ничего общего со специальным объектом временной шкалы.

ВАЖНО: к объекту анимации **нельзя** прикреплять скриптовый объект (иначе отобразится сообщение об ошибке, которое невозможно закрыть, кроме как с помощью закрытия всей программы через диспетчер задач), но к его дочерним элементам (также не объектам анимации) **можно**.

Список функций:

Play - воспроизвести анимацию с текущего кадра временной шкалы.
Stop - остановить воспроизведение анимации на текущем кадре временной шкалы.
GotoFrame - перейти к указанному кадру временной шкалы.
GotoAndPlay - перейти к указанному кадру временной шкалы и начать с него воспроизведение анимации.
GotoAndStop - перейти к указанному кадру временной шкалы и остановить на нём воспроизведение анимации.

Play

Воспроизвести объект анимации с текущего кадра временной шкалы.

Синтаксис:

Play()

Пример:

Tween1.Play()

Stop

Остановить воспроизведение объекта анимации на текущем кадре временной шкалы.

Синтаксис:

Stop()

Пример:

Tween1.Stop()

GotoFrame

Перейти к указанному кадру временной шкалы объекта анимации.
Если анимация воспроизводилась, то воспроизведение продолжится с данного кадра.
Если анимация не воспроизводилась то произойдет переход к статичному изображению данного кадра.

Синтаксис:

GotoFrame(frameNumber)

Параметры:

frameNumber - **(Ц)**, номер кадра временной шкалы (нумерация начинается с **0**). **ОП**.

Пример:

Tween1.GotoFrame(25)

GotoAndPlay

Перейти к указанному кадру временной шкалы объекта анимации и начать с него воспроизведение.

Синтаксис:

GotoAndPlay(frameNumber)

Параметры:

frameNumber - **(Ц)**, номер кадра временной шкалы (нумерация начинается с **0**). **ОП**.

Пример:

Tween1.GotoAndPlay(25)

GotoAndStop

Перейти к указанному кадру временной шкалы объекта анимации и остановить на нём воспроизведение.

Синтаксис:

GotoAndStop(frameNumber)

Параметры:

frameNumber - **(Ц)**, номер кадра временной шкалы (нумерация начинается с **0**). **ОП**.

Пример:

Tween1.GotoAndStop(25)

Таймлайны (временные шкалы)

Временная шкала - последовательность, выполняемых в определенные отрезки времени, действий (**Actions**) с различными объектами. Используется, например, для упрощенного создания комплексных анимаций. Чтобы временная шкала автоматически не запускалась, нужно правой кнопкой на объекте временной шкалы выбрать **"Edit Timeline"**, в открывшемся окне нажать кнопку **"Properties"**, и убрать галочку с опции **"Auto Start"**.

Список функций:

Start - запустить воспроизведение временной шкалы.
Stop - остановить воспроизведение временной шкалы.

Start

Начать воспроизведение временной шкалы.

Синтаксис:

Start()

Пример:

Timeline1.Start()

Stop

Остановить воспроизведение временной шкалы. После остановки временную шкалу снова запустить нельзя.

Синтаксис:

Stop()

Пример:

Timeline1.Stop()

Видео

Видеообъект содержит видеофайл поддерживаемого формата: **GIF**, **FLC**, **FLI**, **MNG**, **AVI**, **MPG**, **MPEG**, **M1V**, **VOB**, **MOV**, **MP4**, **ASF**, **WMA**, **WMV**, **SWF**, **FLV**. Для воспроизведения большинства видеоформатов требуются установленные кодеки. Функции видеообъектов используются для управления воспроизведением видео в публикации. Пока видео играет, следующие строчки скрипта выполняются.

Список функций:

Play - воспроизвести видео.
Stop - остановить воспроизведение видео.
IsPlaying - проверить воспроизводится ли видео.
Go - установить стартовую позицию воспроизведения видео.
Seek - перемотать видео в определенную позицию.
GetLength - получить длину видео в секундах.
GetLengthBytes - получить длину видео в байтах.
GetBufferLength - получить длину видеобуфера.
GetPosition - получить текущую позицию воспроизведения видео в секундах.

Play

Воспроизвести видео или продолжить воспроизведение остановленного видео.

Синтаксис:

Play(From, To, Synch)

Параметры:

From - **(Ч)**, позиция в секундах, с которой начнется воспроизведение видео. **НП**.
To - **(Ч)**, позиция в секундах, на которой закончится воспроизведение видео. При значении **-1** видео проиграется до конца. **НП**.
Если оба параметра **From** и **To** не указаны, воспроизводится всё видео целиком.
Synch - **(Б)**, отправка сообщения о синхронизации (**true** - отправить, **false** - не отправлять). **НП**, по умолчанию **true**.

Stop

Остановить воспроизведение видео.

Синтаксис:

Stop(Reset, Synch)

Параметры:

Reset - **(Б)**, сброс видео (**true** - сброс к началу, **false** - остановка на текущем моменте). **НП**, по умолчанию **false**.
Synch - **(Б)**, отправка сообщения о синхронизации (**true** - отправить, **false** - не отправлять). **НП**, по умолчанию **true**.

IsPlaying

Проверить, воспроизводится ли видео в данный момент или нет.

Синтаксис:

IsPlaying()

Возврат:

(Б), состояние воспроизведения видео (**true** - воспроизводится, **false** - нет).

Общий пример:

```
Video1.Play() //запуск видео с самого начала
Debug.trace("Играет ли видео? "+Video1.IsPlaying())
wait(3);Video1.Stop() //остановить видео через три секунды после запуска
Debug.trace("\n"+"Играет ли видео? "+Video1.IsPlaying())
wait(3);Video1.Play() //продолжить видео через три секунды с места остановки
Debug.trace("\n"+"Играет ли видео? "+Video1.IsPlaying())
```

Go

Заранее установить позицию начала воспроизведения видео, до проигрывания. Используется перед функцией **Play**, в которой не указан параметр начальной позиции воспроизведения. Если же данный параметр указан, то он заменит начальную позицию воспроизведения, стоящую в функции **Go**.

Синтаксис:

Go(Position)

Параметры:

Position - **(Ч)**, позиция в секундах, с которой начнется воспроизведение видео (**-1** для конца видео). **ОП**.

Пример:

```
Video1.Go(3) //установить стартовую позицию воспроизведения для видео
Video1.Play() //начать воспроизведение видео не с начала, а с третьей секунды
```

Seek

Перемотать видео в определенную позицию.

Синтаксис:

Seek(Action, Amount)

Параметры:

Action - **(С)**, способ перемотки, возможные значения: **"forward"** (перемотать вперед), **"backward"** (перемотать назад), **"end"** (перемотать в конец), **"start"** (перемотать в начало). **ОП**.

Amount - **(Ч)**, время в секундах на сколько следует перемотать видео вперед или назад (используется только для способов перемотки: **"forward"** и **"backward"**).

Пример:

```
Video1.Seek("forward", 3.987) //перемотать видео вперед на указанное количество секунд
Video1.Seek("end") //перемотать видео в самый конец
```

GetLength

Получить длину видео в секундах.

Синтаксис:

GetLength()

Возврат:

(Ч), длина видео в секундах (**-1** если не определилась).

Пример:

```
Debug.trace("Длина видео: "+Video1.GetLength()+" сек.")
```

GetLengthBytes и GetBufferLength

Получить длину видео в байтах и длину видеобуфера.

Синтаксис:

GetLengthBytes()
GetBufferLength()

Возврат:

(Ч), длина видео в байтах и длина видеобуфера (**-1** если не определилась).

Пример:

```
Debug.trace("Длина видео в байтах: "+Video1.GetLengthBytes())
Debug.trace("\n"+"Длина видеобуфера: "+Video1.GetLengthBytes())
```

GetPosition

Получить текущую временную позицию воспроизведения видео. Замещает обычное действие функции получения координат расположения объекта (чтобы узнать координаты видеообъекта, используются функции **GetXPosition** и **GetYPosition**).

Синтаксис:

GetPosition()

Возврат:

(Ч), текущая временная позиция воспроизведения видео в секундах.

Пример:

```
Debug.trace("Текущая позиция: "+Video1.GetPosition()+" сек.")
```

Звук и музыка

Функции звука и музыки используются для воспроизведения в публикации музыкальных **CD** и звуковых файлов поддерживаемых форматов: **WAV**, **MP3**, **OGG**, **MID**, **RMI**, **WMA**, **WMV**, **ASF**.

Список функций:

PlaySystemSound - воспроизвести системный звук.
OpenSound - открыть звуковой файл и создать звуковой объект без воспроизведения.
CreateCDPlayer - создать объект **CD**-плеера.
PlaySound - воспроизвести звуковой файл.
PlayCDTrack - воспроизвести трек с музыкального **CD**.
Play - воспроизвести звуковой объект или объект **CD**-плеера.
IsPlaying - проверить воспроизводится ли объект **CD**-плеера.
GetTrackLength - получить длину **CD**-трека в секундах.
Stop - остановить воспроизведение звукового объекта или объекта **CD**-плеера.
GetPosition - получить текущую позицию воспроизведения звукового объекта в секундах.
SetPosition - установить текущую позицию воспроизведения звукового объекта в секундах.
Seek - перемотать звуковой объект в определенную позицию.
Pause - приостановить воспроизведение звукового объекта (только **WAV** и **MP3**) в текущей позиции.
Resume - возобновить воспроизведение звукового объекта (только **WAV** и **MP3**) с места приостановки.
GetVolume - получить текущий уровень громкости устройства или звукового объекта.
SetVolume - установить уровень громкости устройства или звукового объекта.

PlaySystemSound

Воспроизвести системный звук.

Синтаксис:

PlaySystemSound(Type)

Параметры:

Type - **(С)**, тип системного звука, возможные значения: **"Default"**, **"Asterisk"**, **"Question"**, **"Exclamation"**, **"Error"**. **ОП**.

Пример:

```
PlaySystemSound("Error") //воспроизвести системный звук об ошибке
```

OpenSound

Открыть звуковой файл без воспроизведения и сделать его звуковым объектом.

Синтаксис:

OpenSound(Sound, Channel)

Возврат:

(О), звуковой.

Параметры:

Sound - **(С)**, путь к звуковому файлу. **ОП**.
Channel - **(Ц)** от **1** до **8**, канал для воспроизведения звука. **(С) "any"** для воспроизведения звука любым каналом. **НП**, по умолчанию **"any"**.

Пример:

```
var Sound1=OpenSound(SYSTEM_PUBLICATION_DIR+"MUSIC.MP3")
```

CreateCDPlayer

Открыть музыкальный **CD** без воспроизведения и сделать его объектом **CD**-плеера.

Синтаксис:

CreateCDPlayer()

Возврат:

(О) CD-плеера.

Пример:

```
var CDMusic1=CreateCDPlayer()
```

PlaySound, PlayCDTrack и Play

Функция **Playsound** воспроизводит напрямую звуковой файл, без функции **OpenSound**.
Функция **PlayCDTrack** воспроизводит напрямую трек музыкального **CD**, без функции **CreateCDPlayer**.
Функция **Play** воспроизводит либо звуковой объект, созданный функцией **OpenSound**, либо трек объекта **CD**-плеера, созданного функцией **CreateCDPlayer**.

Синтаксис:

PlaySound(Sound, Preload, Times, Volume, Start, Finish, FadeIn, FadeOut, Stop, Channel, Wait)
PlayCDTrack(Track, Times, Volume, Start, Finish, FadeIn, FadeOut, Stop, Wait)
Для звукового объекта:
Play(Times, Volume, Start, Finish, FadeIn, FadeOut, Stop, Wait)
Для объекта CD-плеера:
Play(Track, Times, Volume, Start, Finish)

Параметры:

Sound - **(С)**, путь к звуковому файлу. **ОП**.
Track - **(Ц)**, номер **CD**-трека для воспроизведения (нумерация начинается с **1**). **ОП**.
Preload - **(Б)**, предварительная загрузка звука (**true** - да, **false** - нет). **НП**, по умолчанию **false**.
Times - **(Ц)**, количество раз воспроизведения звука/трека (**-1** для циклического воспроизведения). **НП**, по умолчанию **1**.
Volume - **(Ц)** от **0** до **100**, процентная громкость воспроизведения звука/трека. **НП**, по умолчанию **100**.
Start - **(Ч)**, позиция начала воспроизведения звука/трека в секундах. **НП**, по умолчанию **0** (с самого начала).
Finish - **(Ч)**, позиция конца воспроизведения звука/трека в секундах. **НП**, по умолчанию **-1** (до самого конца).
FadeIn - **(Ч)**, длина в секундах с начала, для приглушения звука на данном участке. **НП**, по умолчанию **0** (без приглушения).
FadeOut - **(Ч)**, длина в секундах с конца, для приглушения звука на данном участке. **НП**, по умолчанию **0** (без приглушения).
Stop - **(Б)**, остановка воспроизведения звука/трека при смене страницы (**true** - остановить, **false** - нет). **НП**, по умолчанию **true**.
Channel - **(Ц)** от **1** до **8**, канал воспроизведения звука. **(С)** **"any"** для воспроизведения звука любым каналом. **НП**, по умолчанию **"any"**.
Wait - **(Б)**, выполнение следующей строчки кода (**true** - только после завершения воспроизведения звука/трека, **false** - сразу после старта воспроизведения звука/трека). **НП**, по умолчанию **true** для звуковых объектов и **false** для **CD**-треков.

Пример:

```
PlaySound(SYSTEM_PUBLICATION_DIR+"MUSIC.MP3") //воспроизвести звуковой файл
PlayCDTrack(3) //воспроизвести 3-й трек напрямую с музыкального компакт диска
var Sound1=OpenSound(SYSTEM_PUBLICATION_DIR+"MUSIC.MP3") //создать звуковой объект
Sound1.Play() //воспроизвести звуковой объект
var CDMusic1=CreateCDPlayer() //создать объект CD-плеера
CDMusic1.Play(3) //воспроизвести 3-й трек объекта CD-плеера
```

IsPlaying

Проверить, воспроизводится ли указанный объект **CD**-плеера в данный момент или нет.

Синтаксис:

IsPlaying()

Возврат:

(Б), состояние воспроизведения объекта **CD**-плеера (**true** - воспроизводится, **false** - нет).

Пример:

```
var CDMusic1=CreateCDPlayer();CDMusic1.Play(1)
Debug.trace("Проигрывается ли CD? "+CDMusic1.IsPlaying())
```

GetTrackLength

Получить длину **CD**-трека.

Синтаксис:

GetTrackLength(Track)

Возврат:

(Ч), длина **CD**-трека в секундах. Если трека под указанным номером не существует, либо **CD** отсутствует в приводе, то **-1**.

Параметры:

Track - **(Ц)**, номер **CD**-трека (нумерация начинается с **1**). **ОП**.

Пример:

```
var CDMusic1=CreateCDPlayer()
Debug.trace("Длительность 1-го CD-трека: "+CDMusic1.GetTrackLength(1)+" сек.")
```

Stop

Остановить и сбросить воспроизведение звукового объекта, либо остановить воспроизведение объекта **CD** -плеера.

Синтаксис:

Stop(Reset)

Параметры:

Reset - **(Б)**, последующее воспроизведение данного звукового объекта (**true** - с самого начала, **false** - с места остановки). **ОП** для звукового объекта, а для объекта **CD**-плеера не указывается.

Пример:

```
var Sound1=OpenSound(SYSTEM_PUBLICATION_DIR+"MUSIC.MP3");Sound1.Play()
wait(30);Sound1.Stop(false) //остановить воспроизведение звука через 30 секунд без сброса
wait(10);Sound1.Play() //возобновить воспроизведение звука через 10 секунд с места остановки
```

GetPosition

Получить текущую временную позицию воспроизведения звукового объекта.

Синтаксис:

GetPosition()

Возврат:

(Ч), текущая временная позиция воспроизведения звукового объекта в секундах.

Пример:

```
var Sound1=OpenSound(SYSTEM_PUBLICATION_DIR+"MUSIC.MP3");Sound1.Play()
wait(30);Debug.trace("Позиция: "+Sound1.GetPosition())
```

SetPosition

Установить временную позицию воспроизведения звукового объекта.

Синтаксис:

SetPosition(From)

Параметры:

From - **(Ч)**, временная позиция в секундах, с которой будет воспроизводиться звуковой объект. **ОП**.

Пример:

```
var Sound1=OpenSound(SYSTEM_PUBLICATION_DIR+"MUSIC.MP3")
Sound1.SetPosition(30) //установить позицию старта воспроизведения на 30 секунду
Sound1.Play() //воспроизвести звуковой объект с 30 секунды
```

Seek

Перемотать звуковой объект в определенную позицию.

Синтаксис:

Seek(Action, Amount)

Параметры:

Action - **(С)**, способ перемотки, возможные значения: **"forward"** (перемотать вперед), **"backward"** (перемотать назад), **"end"** (перемотать в конец), **"start"** (перемотать в начало). **ОП**.

Amount - **(Ч)**, время в секундах, на сколько следует перемотать звуковой объект вперед или назад (используется только для способов перемотки: **"forward"** и **"backward"**).

Пример:

```
var Sound1=OpenSound(SYSTEM_PUBLICATION_DIR+"MUSIC.MP3");Sound1.Play()
wait(10);Sound1.Stop(false) //остановить воспроизведение звука через 10 секунд без сброса
Sound1.Seek("forward", 30.5) //перемотка на 30.5 секунд вперед с места остановки
Sound1.Play() //воспроизведение звука с места окончания перемотки
```

Pause

Приостановить воспроизведение звукового объекта.

Функция предназначена только для форматов **WAV** и **MP3** (с **OGG** и **MID** **не работает**).

Синтаксис:

Pause()

Пример:

```
var Sound1=OpenSound(SYSTEM_PUBLICATION_DIR+"MUSIC.WAV");Sound1.Play()
wait(10);Sound1.Pause() //приостановить воспроизведение звука через 10 секунд
wait(10);Sound1.Resume() //возобновить воспроизведение звука через 10 секунд с места паузы
```

Resume

Возобновить воспроизведение звукового объекта.

Функция предназначена только для форматов **WAV** и **MP3** (с **OGG** и **MID** **не работает**).

Синтаксис:

Resume()

Пример:

```
var Sound1=OpenSound(SYSTEM_PUBLICATION_DIR+"MUSIC.WAV");Sound1.Play()
wait(10);Sound1.Pause() //приостановить воспроизведение звука через 10 секунд
wait(10);Sound1.Resume() //возобновить воспроизведение звука через 10 секунд с места паузы
```

GetVolume

Получить текущий уровень громкости устройства или звукового объекта.

Синтаксис:

Для устройства:

GetVolume(Device, Channel)

Для звукового объекта:

GetVolume()

Возврат:

(Ц), текущий уровень громкости устройства или звукового объекта, в процентах.

Параметры:

Device - **(С)**, устройство, проверяемое на громкость, возможные значения: **"Wave"**, **"Midi"**, **"CD"**. **ОП**.

Channel - **(Ц)** от **1** до **8**, канал (только устройства **"Wave"**), проверяемый на громкость. **НП**, по умолчанию **-1** (все звуки на устройстве **"Wave"**).

SetVolume

Установить уровень громкости устройства или звукового объекта.

Уровень громкости звукового объекта следует изменять после его воспроизведения, т.к. при воспроизведении уже задается громкость.

Синтаксис:

Для устройства:

SetVolume(Device, Volume, Fade, Channel)

Для звукового объекта:

SetVolume(Volume, Fade)

Параметры:

Device - **(С)**, устройство, для изменения громкости, возможные значения: **"Wave"**, **"Midi"**, **"CD"**. **ОП**.

Volume - **(Ц)** от **0** до **100**, процентная громкость для устройства или звукового объекта. **ОП**.

Fade - **(Ч)**, время в секундах за которое должна измениться громкость. **НП**, по умолчанию **0** (резкая смена громкости без перехода).

Channel - **(Ц)** от **1** до **8**, канал (только для устройства **"Wave"**), на котором изменится громкость звуков. **НП**, по умолчанию **-1** (все звуки на устройстве **"Wave"**).

Общий пример:

```
var Sound1=OpenSound(SYSTEM_PUBLICATION_DIR+"MUSIC.WAV");Sound1.Play()
Debug.trace("Начальная громкость звука: "+Sound1.GetVolume())
Debug.trace("\n"+"Начальная громкость устройства: "+GetVolume("Wave"))
wait(10);Debug.trace("\n"+"Громкость звука уменьшается")
Sound1.SetVolume(50, 10);wait(12) //изменить громкость звукового объекта
Debug.trace("\n"+"Новая громкость звука: "+Sound1.GetVolume())
wait(10);Debug.trace("\n"+"Громкость устройства уменьшается")
SetVolume("Wave", 5, 10);wait(12) //изменить громкость звукового устройства
Debug.trace("\n"+"Новая громкость устройства: "+GetVolume("Wave"))
Debug.trace("\n"+"Финальная громкость звука: "+Sound1.GetVolume())
```

QuickTime Virtual Reality (QTVR)

Функции **QTVR**-объектов предназначены для управления в публикации интерактивными панорамами поддерживаемых форматов: **MOV**, **MP4**. Для работы **QTVR**-объектов в системе должен быть установлен **QuickTime Player 6** (с версией **7** некоторые панорамы не отображаются). Некоторые функции либо не работают, либо работают с ошибками. Также создание из панорамных изображений собственных интерактивных панорам с узлами и активными зонами требует дополнительного программного обеспечения, поэтому легче сделать свою интерактивную панораму на основе панорамного изображения с помощью функций графических объектов, не используя **QTVR**-объекты.

Список функций:

GetViewingLimits - получить диапазоны углов и зума **QTVR**-объекта (**не работает**).
GetPanAngle - получить текущий панорамный угол **QTVR**-объекта (в градусах).
SetPanAngle - установить текущий панорамный угол **QTVR**-объекта (в градусах).
GetTiltAngle - получить текущий угол наклона **QTVR**-объекта (в градусах).
SetTiltAngle - установить текущий угол наклона **QTVR**-объекта (в градусах).
GetFieldOfView - получить текущий уровень масштабирования (зум) **QTVR**-объекта.
SetFieldOfView - установить текущий уровень масштабирования (зум) **QTVR**-объекта.
GetCurrentNode - получить идентификатор текущего узла **QTVR**-объекта.
GetVisibleHotspots - получить массив идентификаторов активных зон узла **QTVR**-объекта (**не работает**).
GetHotspotType - получить тип активной зоны текущего узла **QTVR**-объекта.
EnableHotspot - выборочно включить или отключить активные зоны **QTVR**-объекта.
SetMouseOverCallback - установить функцию для наведения мыши на активную зону **QTVR**-объекта.
SetHotspotCallback - установить функцию для щелчка мыши на активной зоне **QTVR**-объекта.

GetViewingLimits

Получить допустимые значения диапазонов углов и масштабирования объекта **QTVR**.

Синтаксис:

GetViewingLimits(Type)

Возврат:

Должен быть: (О) со следующими свойствами:
fMin - **(Ч)**, минимальное допустимое значение для данного диапазона.
fMax - **(Ч)**, максимальное допустимое значение для данного диапазона.
Но из-за **неправильной** работы функции: **null**

Параметры:

Type - **(Ц)**, тип диапазона. **ОП**.
Возможные значения:
0 - панорамный угол.
1 - угол наклона.
2 - уровень масштабирования (зум).

Пример:

```
//Правильно написанный скрипт, но вызывающий ошибку из-за неправильной работы функции
var P1P=P1.GetViewingLimits(0);var P1T=P1.GetViewingLimits(1);var P1Z=P1.GetViewingLimits(2)
Debug.trace("Панорамный угол: MIN="+P1P.fMin+" MAX="+P1P.fMax+"\n")
Debug.trace("Угол наклона: MIN="+P1T.fMin+" MAX="+P1T.fMax+"\n")
Debug.trace("Уровень масштабирования: MIN="+P1Z.fMin+" MAX="+P1Z.fMax)
//Альтернативный рабочий способ получить допустимые значения диапазонов
var TestAngle=360 //максимальный панорамный угол следует находить методом подбора
var P1P=new Object();var P1T=new Object();var P1Z=new Object()
P1.SetPanAngle(0);P1P.fMin=P1.GetPanAngle()
P1.SetPanAngle(TestAngle);P1P.fMax=P1.GetPanAngle()
P1.SetTiltAngle(-90);P1T.fMin=P1.GetTiltAngle()
P1.SetTiltAngle(90);P1T.fMax=P1.GetTiltAngle()
P1.SetFieldOfView(0);P1Z.fMin=P1.GetFieldOfView()
P1.SetFieldOfView(4);P1Z.fMax=P1.GetFieldOfView()
Debug.trace("Панорамный угол: MIN="+P1P.fMin+" MAX="+P1P.fMax+"\n")
Debug.trace("Угол наклона: MIN="+P1T.fMin+" MAX="+P1T.fMax+"\n")
Debug.trace("Уровень масштабирования: MIN="+P1Z.fMin+" MAX="+P1Z.fMax)
```

GetPanAngle

Получить текущий панорамный угол (влево-вправо) **QTVR**-объекта.

Синтаксис:

GetPanAngle()

Возврат:

(Ч), текущий панорамный угол в градусах.

Пример:

```
Debug.trace("Панорамный угол: "+Panorama1.GetPanAngle())
```

SetPanAngle

Установить для **QTVR**-объекта панорамный угол (влево-вправо) из допустимого диапазона.

Синтаксис:

SetPanAngle(Angle)

Параметры:

Angle - **(Ч)**, панорамный угол в градусах (зависит от допустимого диапазона исходника). **ОП**.

Пример:

```
wait(2);Panorama1.SetPanAngle(Panorama1.GetPanAngle()+10)
```

GetTiltAngle

Получить текущий угол наклона (вверх-вниз) **QTVR**-объекта.

Синтаксис:

GetTiltAngle()

Возврат:

(Ч), текущий угол наклона в градусах.

Пример:

```
Debug.trace("Угол наклона: "+Panorama1.GetTiltAngle())
```

SetTiltAngle

Установить для **QTVR**-объекта угол наклона (вверх-вниз) из допустимого диапазона.

Синтаксис:

SetTiltAngle(Angle)

Параметры:

Angle - **(Ч)**, угол наклона в градусах (зависит от допустимого диапазона исходника). **ОП**.

Пример:

```
wait(2);Panorama1.SetTiltAngle(Panorama1.GetTiltAngle()+10)
```

GetFieldOfView

Получить текущий уровень масштабирования (зума) **QTVR**-объекта.

Синтаксис:

GetFieldOfView()

Возврат:

(Ч), текущий уровень масштабирования (зум).

Пример:

```
Debug.trace("Уровень масштабирования: "+Panorama1.GetFieldOfView())
```

SetFieldOfView

Установить для **QTVR**-объекта уровень масштабирования (зума).

Синтаксис:

SetFieldOfView(FOV)

Параметры:

FOV - **(Ч)**, уровень масштабирования (зум), где **0** - максимальное приближение. **ОП**.

Примеры:

```
//Постепенное отдаление, начиная с максимального приближения
for (Zoom=0;Zoom!=1.1;Zoom+=0.02) {Panorama1.SetFieldOfView(Zoom);wait(0.1)}
```

GetCurrentNode

Получить идентификатор текущего узла в **QTVR**-объекте.

Синтаксис:

GetCurrentNode()

Возврат:

(Ц), идентификатор текущего узла (нумерация начинается с **1**).

Пример:

```
Debug.trace("Идентификатор текущего узла: "+Panorama1.GetCurrentNode())
```

GetVisibleHotspots

Получить массив идентификаторов активных зон заданного узла в **QTVR**-объекте.

Синтаксис:

GetVisibleHotspots(ID)

Возврат:

Должен быть: (О), массив идентификаторов видимых активных зон указанного узла.
Но из-за **неправильной** работы функции: пустой массив или ошибка.

Параметры:

ID - **(Ц)**, идентификатор узла (нумерация начинается с **1**). **ОП**.

Пример:

```
//Правильно написанный скрипт, но вызывающий ошибку из-за неправильной работы функции
var HSArray=Panorama1.GetVisibleHotspots(Panorama1.GetCurrentNode())
Debug.trace("Идентификатор первой активной зоны текущего узла: "+HSArray[0])
```

GetHotspotType

Получить тип заданной активной зоны текущего узла в **QTVR**-объекте.

Синтаксис:

GetHotspotType(ID)

Возврат:

(Ц), тип указанной активной зоны текущего узла, возможные значения:
1 - активная зона перехода на другой узел.
2 - активная зона с **URL**-ссылкой.
4 - неопределенная активная зона (активной зоны с указанным идентификатором не существует).

Параметры:

ID - **(Ц)**, идентификатор активной зоны текущего узла (нумерация начинается с **1**). **ОП**.

Пример:

```
Debug.trace("Тип активной зоны: "+Panorama1.GetHotspotType(1))
```

EnableHotspot

Выборочно включить/отключить активные зоны **QTVR**-объекта. Функция работает **неправильно** и действует на **все** активные зоны, независимо от введённых параметров.

Синтаксис:

EnableHotspot(Type, ID, Enable)

Параметры:

Type - **(Ц)**, активная зона для включения/отключения. **ОП**.
Возможные значения:
1 - конкретная активная зона с заданным идентификатором.
2 - все активные зоны указанного типа.
3 - все активные зоны.
ID - **(Ц)**, идентификатор активной зоны (нумерация начинается с **1**), если значение **Type** равно **1**. **ОП**.
Если **Type** равно **2**, то возможные значения:
1 - активные зоны перехода на другой узел.
2 - активные зоны с **URL**-ссылками.
4 - неопределенные активные зоны.
Если **Type** равно **3**, то **0**.
Enable - **(Б)**, состояние работы активных зон (**true** - включить, **false** - отключить). **ОП**.

Пример:

```
Panorama1.EnableHotspot(3, 0, false) //отключить все активные зоны QTVR-объекта
```

SetMouseOverCallback

Установить функцию, вызываемую при наведении курсора мыши на любую активную зону **QTVR**-объекта.

Синтаксис:

SetMouseOverCallback(MOHSCallback)

Параметры:

MOHSCallback - название пользовательской функции с одним параметром, вызываемой при наведении курсора мыши на любую активную зону **QTVR**-объекта. **ОП**.

Параметр функции будет объектом со следующими свойствами:

nID - **(Ц)**, идентификатор активной зоны (нумерация начинается с **1**), на которую наведен курсор мыши.

bEntering - **(Б)**, состояние наведения курсора мыши (**true** - курсор мыши был наведен на активную зону, **false** - уже наведенный курсор был убран с активной зоны).

Пример:

```
function MouseOverQTVR(ObjHSMO) //объявить пользовательскую функцию
{if (ObjHSMO.bEntering==true) {Debug.trace("Мышь над активной зоной № "+ObjHSMO.nID+"\n")}}
//назначить функцию QTVR-объекту, выполняемую при наведении мышью на активную зону
Panorama1.SetMouseOverCallback(MouseOverQTVR)
```

SetHotspotCallback

Установить функцию, вызываемую при щелчке мышью на любой активной зоне **QTVR**-объекта.

Синтаксис:

SetHotspotCallback(HSCallback)

Параметры:

HSCallback - название пользовательской функции с одним параметром, вызываемой при нажатии мышью на любую активную зону **QTVR**-объекта. **ОП**.

Параметр функции будет объектом со следующими свойствами:

nID - **(Ц)**, идентификатор активной зоны (нумерация начинается с **1**), на которой была нажата мышь.

nNodeID - **(Ц)**, идентификатор узла, в котором была нажата мышь на активной зоне.

Пример:

```
function MouseClickQTVR(ObjHSMC) //объявить пользовательскую функцию
{Debug.trace("Идентификатор узла: "+ObjHSMC.nNodeID+"\n")
Debug.trace("Идентификатор активной зоны: "+ObjHSMC.nID+"\n")}
//назначить функцию QTVR-объекту, выполняемую при нажатии мышью на активную зону
Panorama1.SetHotspotCallback(MouseClickQTVR)
```

Базы данных

Для работы с базами данных необходимы минимальные знания **SQL** (**Structured Query Language**).
Для использований функций базы данных сперва следует **создать новый объект базы данных** в скрипте, который ссылается на название источника данных **DSN** (**Data Source Name**).
Если **DSN**-файл находится на диске:
var DB1=new Database("FILEDSN="+Path+";")
Path - **(C)**, полный путь к **DSN**-файлу на диске, например: **"C:\\OPUSDB\\EXCELDB1.DSN"**
Для **DSN** с путем по умолчанию:
var DB1=new Database("FILEDSN=catalogue.dsn;")
Для машин **DSN**:
var DB1=new Database("DSN=catalogue;")

Список функций:
ExecuteSQL - выполнить команду базы данных **SQL**.
Connect - подключиться к серверу **SQL**.
Commit - зафиксировать в базе данных (постоянно хранить в базе данных).
Rollback - вернуться к точке в базе данных до внесения изменений или отправки последней фиксации.
AutoCommitOn - включить или выключить автофиксацию для базы данных.
IsAutoCommitOn - проверить, включена ли автофиксация в базе данных.
GetLastError - получить последнее сообщение об ошибке базы данных.
GetIdentQuoteChar - определить символ кавычки для названий таблиц или полей с пробелами.
GetNumberOfRecords - получить общее количество записей в наборе.
GetCurrentRecordNumber - получить номер текущей записи в наборе.
FirstRecord - перейти к первой записи в наборе.
LastRecord - перейти к последней записи в наборе.
NextRecord - перейти к следующей записи в наборе.
PreviousRecord - перейти к предыдущей записи в наборе.
GetRecordAt - перейти к определенной записи в наборе.
GetRecordAtRelative - перейти к записи в наборе относительно номера текущей.
DATABASE_ERROR_MESSAGE - если в публикации используется база данных, эта переменная будет содержать любые сообщения об ошибках, отправленных через базу данных при возникновении проблемы с подключением или доступом к базе данных.

Для проверки примеров нужно:
1) Создать в блокноте файл **C:\OPUSDB\EXCELDB1.DSN**, содержащий текст:
[ODBC]
DRIVER=Microsoft Excel Driver (*.xls)
UID=admin
UserCommitSync=Yes
Threads=3
SafeTransactions=0
ReadOnly=0
PageTimeout=5
MaxScanRows=8
MaxBufferSize=2048
FIL=excel 8.0
DriverId=790
DefaultDir=C:\OPUSDB
DBQ=C:\OPUSDB\TABLE1.XLS
2) Создать **XLS**-таблицу **C:\OPUSDB\TABLE1.XLS** со следующими значениями:
A1=Z_axis, **B1=Y_axis**, **C1=X_axis**, **D1=Color**, **A2=1**, **B2=2**, **C2=3**, **D2=FF0000**,
A3=63, **B3=31**, **C3=15**, **D3=00FF00**, **A4=8**, **B4=16**, **C4=32**, **D4=0000FF**
Затем выделить диапазон **A1:D4** и присвоить ему название **Coordinates**. Название листа не важно.

ExecuteSQL

Выполнить для базы данных допустимую инструкцию **SQL** (например **SELECT** или **INSERT INTO**).

Синтаксис:

ExecuteSQL(Command)

Возврат:

Для инструкции **SELECT**:

(О) набора записей. **(Б)**, **false**, если произошла ошибка.

Для инструкции **UPDATE** или **INSERT**:

(Б), **true** если действие выполнилось, **false** если не выполнилось.

Параметры:

Command - **(С)**, допустимая инструкция **SQL**. **ОП**.

Пример:

```
var DB1=new Database("FILEDSN=c:\\OPUSDB\\EXCELDB1.dsn;")
var RS1=DB1.ExecuteSQL("SELECT * FROM Coordinates")
```

Connect

Подключиться к серверу **SQL**. Если нет действительного идентификатора пользователя и пароля, то к серверу **SQL** нельзя подключиться.

Синтаксис:

Connect(UserID, Password)

Возврат:

(Б), состояние подключения (**true** - соединение выполнено, иначе **false**).

Параметры:

UserID - **(С)**, действительный идентификатор пользователя **SQL**. **НП**.

Password - **(С)**, действительный пароль для идентификатора пользователя. **НП**.

Система может использовать значения по умолчанию, установленные на вкладке **"Database"** в диалоговом окне **"Publication Properties"**.

Пример:

```
var DB1=new Database("FILEDSN=c:\\OPUSDB\\EXCELDB1.dsn;")
Result=DB1.Connect("Jack", "12345")
```

Commit

Зафиксировать объект в базе данных (запись будет постоянно внесена в базу данных).

Для работы этой функции автофиксация должна быть выключена.

Синтаксис:

Commit()

Возврат:

(Б), фиксация объекта (**true** - объект зафиксирован, иначе **false**).

Пример:

```
var DB1=new Database("FILEDSN=c:\\OPUSDB\\EXCELDB1.dsn;")
DB1.AutoCommitOn(false)
if (DB1.ExecuteSQL("INSERT INTO Coordinates (Z_axis, Y_axis) VALUES (33, 66);"))
{if (DB1.ExecuteSQL("INSERT INTO Coordinates (X_axis, Color) VALUES (6, '111111');"))
{DB1.Commit()} //зафиксировать объект (сохранить изменения)
else {DB1.Rollback()}} //отменить первую вставку, если вторая прошла неудачно
```

RollBack

Вернуться к точке в базе данных до внесения изменений или отправки последней фиксации.

Для работы этой функции автофиксация должна быть выключена.

Синтаксис:

Rollback()

Пример:

```
DB1.Rollback()
```

AutoCommitOn

Включить или выключить автофиксацию. Автофиксация - автоматический ввод данных в базу данных после выполнения инструкций **SQL**: **INSERT** или **UPDATE**. (В оригинальной справке функция названа неправильно: **AutoCommit**).

Синтаксис:

AutoCommitOn(Enable)

Параметры:

Enable - **(Б)**, состояние автофиксации (**true** - включить, **false** - выключить). **НП**, по умолчанию **false**.

Пример:

```
var DB1=new Database("FILEDSN=c:\\OPUSDB\\EXCELDB1.dsn;")
DB1.AutoCommitOn(true)
```

IsAutoCommitOn

Проверить, включена или выключена автофиксация.

Синтаксис:

IsAutoCommitOn()

Возврат:

(Б), состояние автофиксации (**true** - включена, **false** - выключена).

Пример:

```
var DB1=new Database("FILEDSN=c:\\OPUSDB\\EXCELDB1.dsn;")
Debug.trace("Состояние автофиксации: "+DB1.IsAutoCommitOn())
```

GetLastError

Получить (если есть) последнее отправленное сообщение об ошибке базы данных. Если после выполнения **ExecuteSQL** получена ошибка, то можно проверить отправленное сообщение об ошибке.

Синтаксис:

GetLastError()

Возврат:

(С), последнее сообщение об ошибке базы данных.

Пример:

```
var DB1=new Database("FILEDSN=c:\\OPUSDB\\EXCELDB1.dsn;")
var RS1=DB1.ExecuteSQL("SELECT * FROM Points")
Debug.trace("Ошибка: "+DB1.GetLastError())
```

GetIdentQuoteChar

Получить символ кавычка, необходимый при цитировании названий таблиц и полей базы данных, в которых есть пробелы. Использовать его нужно только в том случае, если название таблицы или поля в базе данных содержит пробелы.

Синтаксис:

GetIdentQuoteChar()

Возврат:

(Ц), **ASCII**-код (**96**) символа одинарная кавычка.

Пример:

```
var DB1=new Database("FILEDSN=c:\\OPUSDB\\EXCELDB1.dsn;")
var Quote=DB1.GetIdentQuoteChar()
```

GetNumberOfRecords

Получить количество записей в наборе.

Синтаксис:

GetNumberOfRecords()

Возврат:

(Ц), количество записей в наборе (не количество записей в базе данных).

Пример:

```
var DB1=new Database("FILEDSN=c:\\OPUSDB\\EXCELDB1.dsn;")
var RS1=DB1.ExecuteSQL("SELECT * FROM Coordinates")
Debug.trace("Количество записей: "+RS1.GetNumberOfRecords())
```

GetCurrentRecordNumber

Получить порядковый номер текущей записи набора.

Синтаксис:

```
GetCurrentRecordNumber()
```

Возврат:

(Ц), порядковый номер текущей записи в наборе (нумерация начинается с **1**).

Пример:

```
var DB1=new Database("FILEDSN=c:\\OPUSDB\\EXCELDB1.dsn;")
var RS1=DB1.ExecuteSQL("SELECT * FROM Coordinates")
Debug.trace("Номер записи: "+RS1.GetCurrentRecordNumber())
```

FirstRecord

Перейти к первой записи в наборе.

Синтаксис:

```
FirstRecord()
```

Возврат:

(Б), **true** если записи в наборе существуют, иначе **false**.

Пример:

```
var DB1=new Database("FILEDSN=c:\\OPUSDB\\EXCELDB1.dsn;")
var RS1=DB1.ExecuteSQL("SELECT * FROM Coordinates")
RS1.FirstRecord();Debug.trace("Значение записи: "+RS1.Z_axis)
```

LastRecord

Перейти к последней записи в наборе.

Синтаксис:

```
LastRecord()
```

Возврат:

(Б), **true** если записи в наборе существуют, иначе **false**.

Пример:

```
var DB1=new Database("FILEDSN=c:\\OPUSDB\\EXCELDB1.dsn;")
var RS1=DB1.ExecuteSQL("SELECT * FROM Coordinates")
RS1.LastRecord();Debug.trace("Значение записи: "+RS1.Y_axis)
```

NextRecord

Перейти к следующей записи в наборе.

Синтаксис:

```
NextRecord()
```

Возврат:

(Б), **true** если следующая запись в наборе существует, иначе **false**.

Пример:

```
var DB1=new Database("FILEDSN=c:\\OPUSDB\\EXCELDB1.dsn;")
var RS1=DB1.ExecuteSQL("SELECT * FROM Coordinates")
RS1.NextRecord();Debug.trace("Значение записи: "+RS1.X_axis)
```

PreviousRecord

Перейти к предыдущей записи в наборе.

Синтаксис:

```
PreviousRecord()
```

Возврат:

(Б), **true** если предыдущая запись в наборе существует, иначе **false**.

Пример:

```
var DB1=new Database("FILEDSN=c:\\OPUSDB\\EXCELDB1.dsn;")
var RS1=DB1.ExecuteSQL("SELECT * FROM Coordinates")
RS1.PreviousRecord();Debug.trace("Значение записи: "+RS1.X_axis)
```

GetRecordAt

Перейти к определенной записи в наборе.

Синтаксис:

GetRecordAt(Position)

Возврат:

(Б), **true** если указанная запись в наборе существует, иначе **false**.

Параметры:

Position - **(Ц)**, порядковый номер записи в наборе (нумерация начинается с **1**). **ОП**.

Пример:

```
var DB1=new Database("FILEDSN=c:\\OPUSDB\\EXCELDB1.dsn;")
var RS1=DB1.ExecuteSQL("SELECT * FROM Coordinates")
RS1.GetRecordAt(2);Debug.trace("Значение записи: "+RS1.Color)
```

GetRecordAtRelative

Перейти к записи набора относительно номера текущей записи.

Синтаксис:

GetRecordAtRelative(relativePosition)

Возврат:

(Б), **true** если запись в наборе существует, иначе **false**.

Параметры:

relativePosition - **(Ц)** положительное или отрицательное, на сколько записей относительно номера текущей записи следует перейти. **ОП**.

Пример:

```
var DB1=new Database("FILEDSN=c:\\OPUSDB\\EXCELDB1.dsn;")
var RS1=DB1.ExecuteSQL("SELECT * FROM Coordinates")
RS1.GetRecordAtRelative(-1) //к предыдущей записи
```

Вопросы

Вопрос - особый тип графического объекта кадр с дополнительной вкладкой **"Question"** в свойствах.

Список функций:

GetQuestionCorrect - проверить правильность ответа.
GetQuestionIncorrect - проверить неправильность ответа.
GetQuestionPossible - получить возможное количество баллов за ответ на вопрос.
GetQuestionValue - получить баллы за полученный ответ на вопрос.
SetUserAnswer - указать ответ на вопрос.
ResetUserAnswer - сбросить ответ на вопрос.
Confirm Question - подтвердить ответ на вопрос.
ResetQuestion - сбросить вопрос.

Переменные вопросов

Q_SCORE_VALUE - значение или количество баллов пользователя до сих пор (**8** в **"8 из 10"**). Эта переменная отслеживает общую оценку ответов, которые пользователь получил правильно. Значение для отдельного ответа задается в сетке свойств вопроса.
Q_SCORE_CORRECT - **(Ц)**, количество вопросов, на которые пользователь ответил правильно. В случае вопросов с несколькими ответами для **SCORE_CORRECT** необходимо выбрать только один правильный ответ. Если два ответа помечены как правильные и пользователь выбрал оба ответа, переменные **SCORE_CORRECT** будут по-прежнему увеличиваться только на **1**.
Q_SCORE_INCORRECT - **(Ц)**, количество вопросов, на которые пользователь ответил неверно. В случае нескольких вопросов с ответами пользователю нужно только получить один правильный ответ, чтобы вопрос считался правильным, даже если один из других ответов пользователя обозначен как неправильный.
Q_SCORE_VALUE_POSSIBLE - возможное количество ответов на вопросы до сих пор (**10** в **"8 из 10"**), независимо от того, набрал ли пользователь свои ответы.
Q_SCORE_TOTAL_POSSIBLE - возможное значение всех вопросов во всей публикации (**10** в **"8 из 10"** но для всей публикации).
Q_SCORE_VALUE_TOTAL_REMAINING - общая оценочная стоимость заданий в публикации, на которые еще нет ответа.
Q_SCORE_CURRENTLY_POSSIBLE - отслеживает значение оценки, которую пользователь все еще может получить, если на все оставшиеся вопросы будут даны правильные ответы, включая текущее **Q_SCORE_VALUE**. Позволяет отслеживать, провалил ли пользователь тест, даже до того, как он ответил на все вопросы, то есть его оценка не достигнет порогового значения, даже если с этого момента все получится правильно.
Q_SCORE_PERCENT_POSSIBLE - отслеживает процентную оценку, которую пользователь все еще может получить, если на все оставшиеся вопросы будут даны правильные ответы, включая текущее **Q_SCORE_VALUE_PERCENT**. Позволяет отслеживать, провалился ли пользователь в тесте, даже до того, как он ответил на все вопросы, то есть его оценка не достигнет порогового значения, даже если с этого момента все получится правильно.
Q_SCORE_VALUE_PERCENT - балл, который был присвоен пользователю, в процентах от количества, которое пользователь мог получить к этому моменту.
Q_SCORE_OVERALL_PERCENT - оценка пользователя, присужденная в процентах от оценки, возможной во всем опроснике.
Q_SCORE_ANSWERED_QUESTIONS - **(Ц)**, число вопросов, на которые пользователь пытался ответить до сих пор.
Q_SCORE_TOTAL_QUESTIONS - **(Ц)**, общее количество вопросов во всей публикации. Позволяют отслеживать ход курса и использовать элементы, установленные на вкладке **"Course Settings"**.
Q_SCORE_COURSE_PASSED - **(Б)**, пройден ли курс (**true** - пройден, **false** -нет). **true** только после окончания курса.
Q_SCORE_COURSE_FINISHED - **(Б)**, отслеживает ответы на все вопросы в публикации, закончен ли курс (**true** - закончен, **false** - нет).
Q_SCORE_COURSE_SUMMARY - запись активности учащегося в публикации, которая может быть отображена, распечатана или отправлена по электронной почте.
Q_COURSE_ADMINISTRATOR - **(С)**, имя администратора курса или тренера, как указано в настройках курса в свойствах публикации.
Q_COURSE_ADMINISTRATOR_EMAIL - **(С)**, адрес электронной почты администратора курса, если он указан в настройках курса.

Переменные для темы вопросов

Q_TOPIC_ANSWERED_QUESTIONS - **(Ц)**, число ответов на вопросы.
Q_TOPIC_CORRECT - правильный счет для текущей темы.
Q_TOPIC_INCORRECT - неверный счет для текущей темы.
Q_TOPIC_CURRENT_VALUE - значение пользовательских ответов для активной темы.
Q_TOPIC_CRITICAL_PASS - **(Б)**, если пользователь решает критическое задание неправильно, то **false**.
Q_TOPIC_CURRENT_PERCENT - текущий пользовательский балл по этой теме в процентах.
Q_TOPIC_CURRENT_POSSIBLE - максимум, что пользователь все еще может набрать для этой темы.
Q_TOPIC_CURRENT_VALUE - награды за эту тему.
Q_TOPIC_FINISHED - отслеживает все ли попытки использованы в теме.
Q_TOPIC_NAME - **(С)**, название темы, установленное в свойствах публикации.
Q_TOPIC_PASSED - **(Б)**, проверка, была ли пройдена тема или нет.
Q_TOPIC_PASS_PERCENT - процент, необходимый для прохождения темы, установленный в свойствах публикации.
Q_TOPIC_PASS_VALUE - фактические оценки, необходимые для прохождения темы, установленный в свойствах публикации.
Q_TOPIC_VALUE_POSSIBLE - возможная ценность вопросов до сих пор.
Q_TOPIC_TOTAL_POSSIBLE - общая стоимость оценок, доступных в теме.
Q_TOPIC_VALUE_REMAINING - общая стоимость оставшихся без ответа вопросов.
Q_TOPIC_TOTAL_QUESTIONS - **(Ц)**, общее количество вопросов.
Q_TOPIC_VALUE_TOTAL_REMAINING - отметки, оставленные в оставшихся без ответа вопросах.
Поддерживаются как переменные массива, так что различные темы могут быть доступны. Нумерация массивов всегда начинается с **0** вместо **1**, что может привести к путанице. Таким образом, процентная доля, набранная для первой темы, сохраняется в переменной **TOPIC_VALUE_PERCENT[0]** и неправильные ответы, относящиеся ко второй теме, будут в **TOTAL_INCORRECT[1]**.

Скоринговая информация

SCORE_VALUE - общая стоимость распределенных баллов, на которые были даны правильные ответы. Например, пользователь должен ответить на **4** вопроса с оценками, распределенными таким образом: **Q1=1**, **Q2=3**, **Q3=5**, **Q4=2**. Только первые три ответа верны, поэтому значение оценки будет равно **9**. Если бы все ответы были правильно, счет был бы **11**.
SCORE_VALUE_TOTAL - общее значение распределенных баллов, на которые были даны ответы, независимо от того, правильно ли пользователь ответил на вопрос или нет. В приведенном выше примере, эта переменная будет содержать значение **11**.
SCORE_VALUE_PERCENT - отслеживает общее количество доступных баллов в процентах за один вопрос. Это не общий балл по всем вопросам.
SCORE_CORRECT - используется, чтобы получить число вопросов, на которые были даны правильные ответы, например **5** из **7** вопросов.
SCORE_INCORRECT - используется, чтобы получить число вопросов, на которые были даны неправильные ответы, например **2** из **7** вопросов.
SCORE_TOTAL - число вопросов, на которые пользователь попытался ответить, независимо от правильности ответа. Например, если есть **10** вопросов, пользователь попытался ответить на **7**, но правильно ответил только на **4**, то значение будет равно **7**.
SCORE_PERCENT - процент вопросов, на которые были даны правильные ответы, например **6** правильных ответов из **10** вопросов дают оценку **60**.
SCORE_TOTAL_POSSIBLE - определяемая пользователем переменная, в которой хранится максимально возможная оценка для всего опросника, независимо от того, на сколько вопросов ответил пользователь. Значение добавляется вручную.
SCORE_CURRENT_POSSIBLE - определяемая пользователем переменная для хранения общего возможного балла. Рассчитывается как **SCORE_TOTAL_POSSIBLE-SCORE_VALUE_TOTAL**.
SCORE_PASS_THRESHOLD - определяемая пользователем переменная для хранения оценки, необходимой для достижения прохождения. Это позволяет контролировать значение **SCORE_CURRENT_POSSIBLE** по отношению к отметке прохождения и заставлять пользователя восстанавливать обучение, пока не будет поддержан уровень прохождения.

Имя пользователя и логин

LOGIN_USER_NAME - **(C)**, полное имя пользователя, вошедшего в систему. **SYSTEM_USERNAME** может использоваться как значение по умолчанию, которое пользователь может редактировать, уточнять или расширять, или его можно явно запросить у пользователя.
LOGIN_FIRSTNAME - **(C)**, имя пользователя. Может быть запрошено специально или извлечено из имени пользователя с помощью **QuickScript**.
LOGIN_SURNAME - **(C)**, второе имя пользователя. Может быть запрошено специально или извлечено из имени пользователя с помощью **QuickScript**.
LOGIN_USER_EMAIL - **(C)**, адрес электронной почты пользователя. Для ввода пользователем/дизайнером.
LOGIN_TUTOR - **(C)**, репетитор курса. Для ввода пользователем/дизайнером.
LOGIN_MANAGER - **(C)**, менеджер курса. Для ввода пользователем.
LOGIN_TUTOR_EMAIL - **(C)**, сохранить адрес электронной почты репетитора при необходимости. Для ввода пользователем/дизайнером.
LOGIN_ORGANIZATION - **(C)**, организация лица, выполняющего вход. Для ввода пользователем.
LOGIN_USER_ID - уникальный идентификационный номер для вошедшего в систему пользователя. Может быть получен от пользователя или из базы данных курса или рассчитан изнутри.

Получение значения вопроса

Например есть объект вопроса **Question1**, и надо отобразить обратную связь, сообщающую пользователю, что он набрал в этом конкретном вопросе (а не текущую сумму за весь тест). Можно добавить следующий код к скриптовому объекту в вопросе (используя **this.UserScore**, скрипт создаст переменную объекта вопроса, которую могут видеть другие объекты, что было бы не так, если бы просто использовалась **var** в самом скрипте. В качестве альтернативы можно создать переменную страницы для **UserScore** и возможного рейтинга, и тогда больше не понадобится это в любом из сегментов скрипта ниже.

```
this.UserScore=Question1.GetQuestionValue()
this.PossibleScore=Question1.GetQuestionPossible()
this.CorrectlyAnswered=Question1.GetQuestionCorrect()
```

Затем можно создать фрагмент текста, в котором отображается переменная страницы, которая создана с целью обеспечения обратной связи и включает переменные, например:

Вы набрали <this.UserScore> из <this.PossibleScore>

<this.UserScore> вставляется в текст как **Constant Expression** при использовании параметра **"Insert Variable"** в меню **"Text"**.

Создание обратной связи по мере необходимости

В качестве альтернативы можно адаптировать текст feebdack в зависимости от того, правильно ли ответил пользователь. Когда нажимается кнопка **"Submit"** для вопроса, то проверяется, правильно ли на него был получен ответ с помощью **if**. Затем отображается объект обратной связи: для правильного ответа **correctFeedback**, а для неправильного **incorrectFeedback**.

```
if (CorrectlyAnswered==true)
{correctFeedback.Show()}
else
{incorrectFeedback.Show()}
```

Наконец, может быть один объект **feedbackPanel** с фрагментом текста, отображающим переменную **feedbackText**, и изменяющий **feedbackText** в зависимости от того, правильно ли ответил пользователь. Можно запустить фрагмент скрипта, который оценил ответ и соответствующим образом создал **feedbackText**.

```
if (CorrectlyAnswered==true)
{feedbackText="Отлично, это правильно! Вы набрали "+UserScore+"из "+PossibleScore}
else
{feedbackText="Увы, не правильно! Вы набрали "+UserScore+"из "+PossibleScore}
feedbackPanel.Show()
```

GetQuestionCorrect

Проверить правильность ответа на вопрос.

Синтаксис:

GetQuestionCorrect()

Возврат:

(Б), **true** если **правильно**, иначе **false** (**null** до ответа на вопрос).

GetQuestionIncorrect

Проверить неправильность ответа на вопрос.

Синтаксис:

GetQuestionIncorrect()

Возврат:

(Б), **true** если **неправильно**, иначе **false** (**null** до ответа на вопрос).

GetQuestionPossible

Получить возможное количество баллов за ответ на вопрос .

Синтаксис:

GetQuestionPossible()

Возврат:

(Ц), возможное количество баллов (**N** в **"X из N"**).

GetQuestionValue

Получить баллы за полученный ответ на вопрос.

Синтаксис:

GetQuestionValue()

Возврат:

(Ц), баллы (**null** до ответа на вопрос) за полученный ответ (**X** в **"X из N"**).

SetUserAnswer

Указать ответ на вопрос.

Синтаксис:

SetUserAnswer(AnswerIndex)

Параметры:

AnswerIndex - **(Ц)**, порядковый номер ответа (список в свойствах вопроса, вкладка **"Question"**). **ОП**.

ResetUserAnswer

Сбросить ответ на вопрос для повторной попытки.

Синтаксис:

ResetUserAnswer(AnswerIndex)

Параметры:

AnswerIndex - **(Ц)**, порядковый номер ответа (список в свойствах вопроса, вкладка **"Question"**). **ОП**.

ConfirmQuestion

Подтвердить ответ на вопрос. Оценка и другие переменные вопроса обновляются в соответствии с набором ответов и оценками, указанными в свойствах вопроса. Отдельные ответы не могут быть изменены после этой функции, если предварительно не выполнена **ResetQuestion**.

Синтаксис:

ConfirmQuestion()

ResetQuestion

Сбросить вопрос и отменить действие **ConfirmQuestion**.

Синтаксис:

ResetQuestion()

Интернет и браузеры

Список функций:
LaunchURL - открыть **URL** во внешнем браузере.
InternetGetData - получить информацию с удаленного сервера.
InternetPostData - разместить информацию на удаленном сервере.
SendEmail - отправить электронное письмо (требуется **Outlook**).
Navigate - открыть **URL** в объекте браузера.
Home - перейти на домашнюю страницу в объекте браузера.
Refresh - повторно отобразить текущую страницу в объекте браузера.
Stop - остановить объект браузера.
Back - вернуться назад на одну страницу в истории объекта браузера.
Forward - перейти вперед на одну страницу в истории объекта браузера.
Print - распечатать содержимое объекта браузера.

LaunchURL

Открыть указанный **URL** (унифицированный указатель ресурса) во внешнем браузере.
Синтаксис:
LaunchURL(URL)
Параметры:
URL - **(С)**, **URL** для запуска. **ОП**.
Пример:
```
LaunchURL("www.old-games.ru")
```

InternetGetData

Получить информацию с удаленного сервера.
Синтаксис:
InternetGetData(URL)
Возврат:
Данные с удаленного сервера.
Параметры:
URL - **(С)**, **URL**, указывающий скрипт, который будет оценивать отправленные данные, такие как, **cgi** -скрипт. **ОП**.
Пример:
```
var newData=InternetGetData("www.example.com/cgi- bin/checkdata.pl")
```

InternetPostData

Отправить данные на удаленный сервер и получить ответ.
Синтаксис:
InternetPostData(URL, Data, ReturnObject, Secure)
Параметры:
URL - **(С)**, **URL**, указывающий скрипт, который будет оценивать отправленные данные, такие как, **cgi** -скрипт. **ОП**.
Data - **(С)** или **(О)**, данные для отправки и возврата с сервера. **ОП**.
ReturnObject - **(Б)**, тип возвращаемых данных (**true** - как объект, **false** - как строка). **НП**, по умолчанию **false**.
Secure - **(Б)**, использование защищенного протокола (**true** - **HTTPS**-протокол, **false** - **HTTP**-протокол). **НП**, по умолчанию **false**.
Пример:
```
var URL1="http://www.buka.com/cgi- bin/checkdata.pl"
OUT=InternetPostData(URL1, "data=Opus") //возврат: "data=Opus"
OUT=InternetPostData(URL1, "data=Opus", true) //возврат: объект со свойством data равным "Opus"
//Чтобы отправить объект, его сначала надо настроить
var Obj1=new Object()
Obj1.Game="Vangers";Obj1.Ver=1.5
OUT=InternetPostData(URL1, Obj1, true) //возврат: объект с установленными свойствами Game и Ver
```

SendEmail

Отправить электронное письмо по указанному адресу электронной почты.
ВАЖНО: функция требует установленный **Outlook**.
Синтаксис:
SendEmail(ToList, CC, BCC, Subject, Message, Attachment, Showmail, MAPI)
Параметры:
ToList - **(С)**, адрес электронной почты основного получателя. **ОП**.
CC - **(С)**, адреса электронной почты других получателей, видимых для основного получателя. **ОП**, но может иметь пустую запись **""**.
BCC - **(С)**, адреса электронной почты других получателей, не видимых для основного получателя. **ОП**, но может иметь пустую запись **""**.
Subject - **(С)**, заголовок с темой письма. **ОП**, но может иметь пустую запись **""**.
Message - **(С)**, сообщение для отправки по электронной почте. **ОП**.
Attachment - **(С)**, путь к файлу, прикрепляемого к письму. **ОП**, но может иметь пустую запись **""**.
Showmail - **(Б)**, отображение окна почты перед отправкой (**true** - отобразить, **false** - нет). **НП**, по умолчанию **true**.
MAPI - **(Б)**, отображение окна входа в систему электронной почты перед отправкой (**true** - отобразить, **false** - нет). **НП**, по умолчанию **false**.
Пример:

```
var emText="Hello, John!"
SendEmail("john@digitalworkshop.com", "", "", "Testing", emText)
```

Navigate

Открыть запрошенный **URL** в объекте браузера.
Синтаксис:
Navigate(URL)
Параметры:
URL - **(С)**, **URL** страницы. **ОП**.
Пример:

```
Browser1.Navigate("www.old-games.ru")
```

Home

Перейти на домашнюю страницу в объекте браузера.
Синтаксис:
Home()
Пример:

```
Browser1.Home()
```

Refresh

Обновить текущую страницу, отображаемую в объекте браузера.
Синтаксис:
Refresh()
Пример:

```
Browser1.Refresh()
```

Stop

Остановить объект браузера.
Синтаксис:
Stop()
Пример:

```
Browser1.Stop()
```

Back

Перейти на одну страницу назад в истории объекта браузера.

Синтаксис:

Back()

Пример:

Browser1.Back()

Forward

Перейти на одну страницу вперед в истории объекта браузера.

Синтаксис:

Forward()

Пример:

Browser1.Forward()

Print

Распечатать содержимое объекта браузера или объекта **DocView**.

Синтаксис:

Print(ShowDialog)

Параметры:

ShowDialog - **(Б)**, отображение окна свойств печати перед распечаткой (**true** - отобразить, **false** - нет). **НП**, по умолчанию **false**.

Пример:

Browser1.Print(true)

Внешние DLL

Значительно расширить возможности **Opus Pro** можно с помощью вызова функций из внешних динамических библиотек (**DLL**).
Для изучения функций **WinAPI** рекомендуются:
Справочник **"Microsoft Win32 Programmer's Reference"**
Книга **Charles Petzold "Programming Windows, Fifth Edition"**

Список функций:
LoadDLL - загрузить внешнюю **DLL**.
CallFn - вызвать функцию из внешней **DLL**.

LoadDLL

Загрузить внешнюю **DLL**, чтобы в дальнейшем вызывать из нее функции.
DLL должна экспортировать функции как стандартные недекорированные, использующие соглашение о вызовах языка **C** (**Си**).
Синтаксис:
LoadDLL(Filename)
Возврат:
(O), для доступа к функциям из внешней **DLL**. Если библиотека не может быть загружена, то **null**.
Параметры:
Filename - **(C)**, путь к загружаемому **DLL**-файлу. **ОП**.
Пример:

```
//1) Загрузить внешнюю библиотеку
ExtDLL=LoadDLL(SYSTEM_WINSYS_DIR+"Kernel32.dll")
//2) Если библитека успешно загрузилась, то вызвать функцию из неё
if (ExtDLL) {ExtDLL.CallFn("Beep", false, "slong", "ulong", 2048, "ulong", 1000)}
```

CallFn

Вызвать функцию из загруженной внешней **DLL**.
Если для вызываемой из **DLL** функции требуется сложный тип для параметра (например **структура**), который не поддерживается функцией **CallFn**, то можно написать на знакомом языке программирования собственную вспомогательную **DLL**, которая разделит, неподдерживаемые типы данных на понятные для **OpusScript**.

Синтаксис:

CallFn(Function, HasDisplay, Return, Params <Type, Value …>)

Возврат:

Значение **типа**, указанного в параметре **Return**. Если при вызове функции произошла ошибка, то **-1**. Невозможно получить значение сложного типа (например **структуры**).

Параметры:

Function - **(С)**, точное название вызываемой из **DLL** функции. **ОП**.
HasDisplay - **(Б)**, **true** если вызываемая из **DLL** функция отображает пользовательский интерфейс (он станет активным), иначе **false**. **ОП**.
Return - **(С)**, тип возвращаемого значения из функции (если есть), возможные значения:
"none" - (**void**, **VOID**), ничего.
"uchar" - (**UCHAR**, **bool**) - байт без знака.
"schar" - (**char**, **boolean**, **BOOLEAN**) - байт со знаком.
"ushort" - (**UINT16** , **WORD**) - целое число без знака.
"sshort" - (**short**, **Boolean**) - целое число со знаком.
"ulong" - (**UINT**, **UINT32**, **DWORD**, **HWND**, **HDC**, **HANDLE**, **COLORREF**), длинное целое число без знака.
"slong" - (**int**, **INT32**, **BOOL**, **LONG**, **NULL**), длинное целое число со знаком.
"float" - (**float**, **double**), число с плавающей точкой.
"string" - (**unsigned char pointer**, **unsigned char***), строка. Отвечает за то, чтобы указанная память оставалась действительной после возврата вызванной функции.
Если функция из **DLL** возвращает значение, но оно не требуется, то все равно следует указать его тип для правильного вызова функции.
ВАЖНО: OpusScript не поддерживает значения **без знака**, поэтому они будут преобразованы в значения **со знаком**. Это может привести к ошибкам преобразования для очень больших значений без знака.
Params <...> - параметры, передаваемые в функцию, должны быть указаны в парах **Type, Value**.
(слово **Params** и угловые скобки (шевроны, заключающие в скобки ряд параметров), приведены для ясности и не должны указываться).
Type - **(С)**, тип параметра, возможное значение:
"ulong" - (**UINT**, **UINT32**, **DWORD**, **HWND**, **HDC**, **HANDLE**, **COLORREF**), длинное целое число без знака.
"slong" - (**int**, **INT32**, **BOOL**, **LONG**), длинное целое число со знаком.
"float" - (**float**, **double**), число с плавающей точкой.
"string" - (**unsigned char***), строка с нуль-терминатором в конце.
"hwnd" - (**HWND**), дескриптор окна. Предназначен для создания собственных дисплеев как дочерних элементов окна публикации. **Value** в данном случае игнорируется, поэтому рекомендуется установить его на **0**.
Value - значение параметра.

Пример:

```
var Handle1 //переменная для дескриптора окна
var DC1 //переменная для контекста устройства
var PEN1, PEN2, PEN3 //переменные для кистей
var COL1=RGB(0, 142, 168)
var COL2=RGB(246, 144, 0)
//1) Загрузить внешние DLL
ExtDLL1=LoadDLL(SYSTEM_WINSYS_DIR+"User32.dll")
ExtDLL2=LoadDLL(SYSTEM_WINSYS_DIR+"GDI32.dll")
//2) Получить дескриптор окна и контекст устройства
Handle1=ExtDLL1.CallFn("GetActiveWindow", false, "ulong")
DC1=ExtDLL1.CallFn("GetDC", false, "ulong", "ulong", Handle1)
PEN1=ExtDLL2.CallFn("CreatePen", false, "ulong", "slong", 0, "slong", 4, "ulong", COL1)
PEN2=ExtDLL2.CallFn("CreatePen", false, "ulong", "slong", 0, "slong", 8, "ulong", COL2)
//3) Нарисовать линию
ExtDLL2.CallFn("SelectObject", false, "ulong", "ulong", DC1, "ulong", PEN1)
ExtDLL2.CallFn("MoveToEx", false, "slong", "ulong", DC1, "slong", 20, "slong", 20)
ExtDLL2.CallFn("LineTo", false, "slong", "ulong", DC1, "slong", 640, "slong", 480)
//4) Нарисовать эллипс
ExtDLL2.CallFn("SelectObject", false, "ulong", "ulong", DC1, "ulong", PEN2)
ExtDLL2.CallFn("Ellipse", false, "slong", "ulong", DC1, "slong", 20, "slong", 20, "slong", 40, "slong", 80)
//5) Освободить контекст устройства
ExtDLL1.CallFn("ReleaseDC", false, "slong", "ulong", Handle1, "ulong", DC1)
```

Действия, не имеющие скриптовых аналогов

Категория **Miscellanious**:
Hide Cursor - скрыть курсор мыши.
Show Cursor - отобразить курсор мыши.
Save Screen as Image - сохранить текущее содержимое окна публикации как изображение в формате **PNG** или **JPG**.

Категория **Audio/Video**:
Eject/Close CD tray - выдвинуть/задвинуть лоток диска.

Категория **Window**:
Maximize Window - развернуть окно.
Minimize Window - свернуть окно.
Restore Window - восстановить окно.
Set Window Always on Top - установить режим отображения окна всегда на переднем плане.
Disable Window Always On Top - отключить режим отображения окна всегда на переднем плане.