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.

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.

* 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 * 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")
```

Описание функций

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

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:
числовое **(Ч)**, целочисленное **(Ц)**, логическое (булево) **(Б)**, строковое **(С)**, объект **(О)**.

Параметры:

Раздел, присутствующий в описании, когда у функции есть параметры. К каждому параметру функции дается пояснение и указывается какой тип значения требуется для параметра:
числовое **(Ч)**, целочисленное **(Ц)**, логическое (булево) **(Б)**, строковое **(С)**, объект **(О)**, любое **(Л)**.
При вызове функции вместо конкретного значения параметра, можно использовать переменную, содержащую 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). Некоторые параметры имеют значение по умолчанию, которое используется, если не вводить собственное значение. Если параметр требуется, но имеет значение по умолчанию, можно не вводить это значение, однако, если есть несколько параметров в функции, каждый из которых имеет значение по умолчанию, но необходимо изменить только одно из значений (например третий параметр), то следует включить предыдущие параметры, даже если используются их значения по умолчанию.

Пример:

Пусть есть функция с тремя параметрами **Function(A,B,C)**, где **A** (с числовым значением, по умолчанию равным **13**), **B** (с логическим значением, по умолчанию равным **false**) и **C** (со строковым значением, по умолчанию равным **"tapir"**). Для вызова функции со всеми значениями параметров по умолчанию достаточно ввести **Function()**. Если же надо изменить значение только для параметра **C**, то следует ввести **Function(13,false,"bison")**, а если ввести **Function("bison")**, то функция сработает неправильно, значение **"bison"** присвоится параметру **A**, а параметры **B** и **C** останутся со значениями по умолчанию.

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, данное 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 Krakatuk(VectorName,Amount) //определена функция Krakatuk 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.

```
Krakatuk(Vector1,500) //выполнение функции Krakatuk с указанными значениями параметров
```

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

```
Krakatuk(this,500) //выполнение функции Krakatuk с указанными значениями параметров
```

Определение модифицированной функции без параметров.

```
function SuperKrakatuk() //определена функция SuperKrakatuk 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;SuperKrakatuk()
```

Ключевое слово this

Предназначено для ссылки 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

Позволяет выполнять 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 - **(О)**, 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 RockaRolla() //определена функция RockaRolla без параметров
{this.Spin(180,3,false)
this.Roll(180,3,true)}
function Triplet()  //определена функция Triplet без параметров
{fork(RockaRolla,Vector1);fork(RockaRolla,Vector2);fork(RockaRolla,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

Позволяет 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 - **(С)**, фрагмент кода по содержанию. **ОП**.

Пример:

```
//Результатом будет вывод фразы "Визит к минотавру" в скриптовой консоли
var a="Визит к " //переменной a присвоено строковое значение
var b="Debug." //переменной b присвоено строковое значение
var c="trace" //переменной c присвоено строковое значение
var 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
```

Клавиатура, мышь и джойстик - обзор функций

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.

Функции:

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

Позволяет 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)
var 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

Позволяет активировать пользовательский триггер для указанного объекта.

Синтаксис:

TriggerComponentEvent(CustomTrigger)

Параметры:

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

Пример:

```
Button1.TriggerComponentEvent("Zanzibar")
```

RegisterEventHandler

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

Синтаксис:

RegisterEventHandler(EventName,EventFunction)

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 кавычек), вызываемой при событии. Данная функция может содержать один параметр, который будет названием объекта свойств события. **ОП**.

UnregisterEventHandler

Позволяет удалить уже зарегистрированное событие.

Синтаксис:

UnregisterEventHandler(EventID)

Параметры:

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

Общий пример:

Использование событий клавиатуры:

```
var 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
```

Использование событий мыши:

```
var Event1=RegisterEventHandler("mousewheelup",Anticlockwise)
var Event2=RegisterEventHandler("mousewheeldown",Clockwise)
var 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")}
```

IsKeyPressed

Позволяет 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. Значения нескольких клавиш вводятся через запятую. **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 по отдельности или вместе
} //конец цикла
```

GetKeyState

Позволяет узнать 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

Позволяет включать/выключать 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

Позволяет 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
var 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

Позволяет 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 (значения нескольких кнопок вводятся через запятую). **ОП**.
"Left" (левая кнопка мыши), "Right" (правая кнопка мыши), "Middle" (средняя кнопка мыши).

Примеры:

Проверка нажатия левой кнопки мыши:

```
while (true) {wait(0.1)
if (IsMousePressed("Left")==true) {Debug.trace("Левая кнопка мыши сейчас нажата\n")}}
```

Проверка нажатия хотя бы одной из двух кнопок мыши:

```
while (true) {wait(0.1);if (IsMousePressed("Left","Right")==true)
{Debug.trace("Или левая, или правая или обе эти кнопки мыши сейчас нажаты\n")}}
```

Проверка нажатия сразу двух кнопок мыши одновременно:

```
while (true) {wait(0.1);if  (IsMousePressed("Left")==true &&  IsMousePressed("Right")==true)
{Debug.trace("Левая и правая кнопки мыши сейчас нажаты вместе\n")}}
```

MouseMove

Позволяет 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

Позволяет сымитировать нажатие кнопки мыши.

Синтаксис:

MouseClick(???)

Параметры:

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

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 операторы 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"**, диалоговое окно **"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 - любые 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) //в консоли выводится: 20
Debug.trace(10+""+10) //в консоли выводится: 1010
Debug.trace("Количество зверей: "+100) //в консоли выводится: Количество зверей: 100
```

Вывод значений переменных:

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

Вывод значения выражения:

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

Вывод значений на разных вкладках:

```
var zootype1="Бизон"
var zootype2="Слон"
var zoosum1=9
var zoosum2=5
Debug.tracePane("Animals", zootype1+"\n"+zootype2) //вывод названий зверей в отдельной вкладке
Debug.tracePane("Numbers", zoosum1+"\n"+zoosum2) //вывод чисел в отдельной вкладке
```

Синтаксис разных типов данных

Скрипт поддерживает шесть типов данных: числа, логические (булевы) значения, строки, объекты, 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) //в скриптовой консоли выводится 32987
```

Восьмеричное целое число.

```
Debug.trace(0764) //в скриптовой консоли выводится 500
```

Шестнадцатеричное целое число.

```
Debug.trace(0xFFFF) //в скриптовой консоли выводится 65535
```

Числа с плавающей точкой.

```
Debug.trace(3.1415+"  "+0.25+"  "+.001) //в скриптовой консоли выводится 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.

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

```
var name="Willy Barankin";var code='MPT34M';var book='~"Arabian Nights"~'
Debug.trace(name+"\n"+code+"\n"+ book)
```

Операции со строковыми значениями.

Строковые значения объединяются с помощью знака плюс.

```
var Part1="From the Depths ";var Part2="of Xibalba";Debug.trace(Phrase=Part1+Part2)
```

При сложении строкового значения с числовым возвращается строковое значение.

```
var Word1="Room ";var Number1=1408;Debug.trace(Word1+Number1)  //Результат будет "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) //результатом будет 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") //результатом будет числовое значение 24
Debug.trace("3"*4*"0003"+"\n") //результатом будет числовое значение 36
Debug.trace("36"/4/"3"+"\n") //результатом будет 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") //результатом будет -1.#IND
Debug.trace("\n"+("Word"-"ord").toString()) ////Результатом будет 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")) //на выводе будет: Итог=true
\xHH - позволяет вывести **ASCII**-символ по его коду, где **HH** - двузначное число в шестнадцатиричной системе счисления.
\uHHHH - позволяет вывести **Unicode**-символ по его коду, где **HHHH**-четырехзначное число в шестнадцатиричной системе счисления.
\\ - двойной обратный слеш позволяет использовать символ обратного слеша в строковом значении (**\x5c**). Чаще всего применяется для указания файлового пути.
Debug.trace("c:\\Games\\Soulbringer\\Readme.txt")
\' - позволяет использовать символ одинарных кавычек в строковом значении (**\x27**).
\" - позволяет 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** и **false**. 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

Функции:
isFinite проверить данные на конечность и бесконечность.
typeof получить тип данных.
valueOf преобразовать данные в примитивное значение.
toString преобразовать данные в строковое значение.
parseInt преобразовать данные в целое число.
parseFloat преобразовать данные в число с плавающей точкой.
void получить неопределенное значение.

isFinite

Позволяет 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

Позволяет получить тип данных.
Синтаксис:
typeof(Any)
Параметры:
Any - **(Л)**, проверяемые данные.
Возврат:
(С), название типа данных.
Пример:

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

valueOf

Позволяет преобразовать значения объектов **Number**, **String**, **Boolean**, **Date** в примитивные.

Синтаксис:

valueOf()

Возврат:

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

Пример:

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

Если сравнить одинаковые значения двух объектов одного типа с помощью строгого равенства, эти значения не будут равны, а если преобразовать эти значения в примитивные, то они станут равны.

toString

Позволяет 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:

toString(NumeralSystem)

Возврат:

(С).

Параметры:

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

Пример:

```
var Bool1=new Boolean(1);var Bool2=Bool1.toString() //булевый объект преобразуется в строку
var Numb1=new Number(39);var Numb2=Numb1.toString() //числовой объект преобразуется в строку
var Pi1=Math.PI;var Pi2=Math.PI.toString() //математический объект преобразуется в строку
Debug.trace("Как число: "+(Bool1+Numb1+Pi1)+"\n"+"Как строка:"+(Bool2+Numb2+Pi2)) //результат
var Obj1=Vector1.GetChild(0).toString() //преобразование названия полигона в строку
Debug.trace("\n"+"Название объекта: "+Obj1) //вывод результата в скриптовую консоль
```

parseInt

Позволяет 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
//преобразование строкового значения, начинающегося с 0 без указания системы счисления
Debug.trace(parseInt("0764")+"\n") //100, равносильно parseInt("0x64",16) а цифра 7 игнорируется
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

Позволяет 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,NumeralSystem)

Возврат:

(Ч)

Параметры:

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

NumeralSystem - (Ц), система счисления. **НП**.

Пример:

```
Debug.trace(parseFloat("8.5")+"\n") //8.5
```

void

Позволяет получить неопределенное значение.

Синтаксис:

void(Expression)

Возврат:

undefined

Параметры:

Expression - (Л). **ОП**.

Пример:

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

Синтаксис операторов

Приоритет операторов.

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

Арифметические операторы.

+ 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)  //вывод в консоли: 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)**

Операторы присваивания.

Такие операторы присваивают значение левой стороне от знака на основе значения правой стороны от знака. **ОЧЕНЬ ВАЖНО** запомнить разницу между **оператором присваивания** = и **оператором 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**

Операторы 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")+" "+("2A"=="2A")) //true true true**
=== строгое равно, например: **Debug.trace((2===2)+" "+(2==="2")+" "+("2A"==="2A")) //true false true**
!= не равно, например: **Debug.trace((3!=2)+" "+("lion"!="cat")+" "+("c"+"at"!="cat")) //true true false**
< 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>=1)) //true true**

Строковые операторы.

+ объединяет (сцепляет) строки, например: **x="I "+"can't "+"go "+"on"** равносильно **x="I can't go on"**
+= добавляет к строке, например: **x="Prisoner Of ";x+="Your Eyes"** равносильно **x="Prisoner Of Your Eyes"**

НЕЛЬЗЯ разъединять строковые значения и отбавлять от них с помощью операторов и -=.

Сравнение строковых значений с помощью операторов < 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 операторы.

Используются для проверки нескольких выражений, возвращают булево значение.

&& логическое **И (AND)**, возвращает **true** если **каждое** выражение истинно, иначе **false**, например:

```
var 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:

```
var x=0, y=0, z=1;Debug.trace(x==0 || y==0 || z==0) //true
```

^ логическое **И**сключающее **ИЛИ (XOR)**, возвращает **1** если **только одно** выражение истинно, иначе **0**, например:

```
var 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 операторы.

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 **И**сключающее **ИЛИ (XOR)**, например: `Debug.trace(11^13) //6 (1011 XOR 1101 = 0110)`

~ побитовое **НЕ (NOT)** - **не работает**.

При использовании операторов 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

Тернарный оператор.

?: позволяет присвоить 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) //вывод результата в скриптовой консоли
```

Оператор разделения команд.

; (точка с запятой) - разделитель команд в коде, например: `a=0;b=1`

После каждой команды кода должна стоять точка с запятой для отделения одной команды от другой. Необязательно использовать разделитель если команды расположены на разных строчках. Если же писать команды в одну строчку то использование разделителя обязательно.

Оператор последовательного вычисления:

, (запятая) позволяет последовательно вычислять значения выражений в скобках, например:

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

Синтаксис переменных

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

Синтаксис:

var variableName=value

Пояснение:

var - оператор 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 оператора **var**, то она становится свойством страницы и может использоваться вне функции или в другой функции.

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

4) Все переменные вне функций являются свойствами страницы.

variableName - название переменной, заданное пользователем.

value - значение переменной.

1) Название переменной должно состоять только из латинских букв и цифр. Допустимыми специальными символами в названии являются: подчёркивание (**_**) и доллар (**$**).

2) Название переменной должно начинаться только с буквенного символа, а не с цифры.

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** и **Amount_01**.

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

Примеры:

***Объявление переменной с инициализацией без оператора* var.**

Variable0=120

Объявление переменной.

var Variable1

Объявление нескольких переменных.

var Variable1, Variable2, Variable3

Инициализация объявленнной переменной.

Variable1=45

Инициализация нескольких объявленных переменных.

Variable1=30, Variable1=60, Variable1=90

Объявление переменных вместе с инициализацией.

```
var w="В итоге: " //объявленной переменной присваивается строковое значение
var x=65 //объявленной переменной присваивается числовое значение
var y=x+35 //объявленной переменной присваивается числовое значение выражения
var z=w+y //объявленной переменной присваивается строковое значение выражения
Debug.trace(z) //вывод значения последней переменной в скриптовую консоль
```

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 - **(C)**, расположение текущей публикации (**null**, если публикация не сохранена).
SYSTEM_WIN_DIR - **(C)**, расположение директории **Windows**, установленной на компьютере.
SYSTEM_WINSYS_DIR - **(C)**, расположение системной директории установленной **Windows**.
SYSTEM_PROGRAMS_DIR - **(C)**, расположение директории **Program Files**.
SYSTEM_PROGRAMDATA_DIR - **(C)**, расположение директории **Program Data** (в **Vista** и выше).
SYSTEM_DOCUMENTS_DIR - **(C)**, расположение директории **Мои документы** текущего пользователя **Windows**.
SYSTEM_TEMP_DIR - **(C)**, расположение директории **Temp** текущего пользователя **Windows**.
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 - **(C)**, 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 - **(C)**, имя текущего пользователя **Windows**.
SYSTEM_OPERATING_SYS - **(C)**, тип текущей операционной системы **Windows**. Возможные значения: **NT3**, **95**, **98**, **ME**, **NT4**, **2000**, **XP**, **2003**, **Vista**, **Windows 7**.

Переменные системной даты и времени:

SYSTEM_DATE_FULL - **(C)**, полная текущая системная дата (день, месяц, год).
SYSTEM_TIME_YEAR - **(Ц)**, текущий год (четырехзначный) системной даты.
SYSTEM_TIME_MONTH - **(C)**, текущий месяц (трехбуквенный) системной даты.
SYSTEM_TIME_DATE - **(C)**, текущее число (день месяца) системной даты.
SYSTEM_TIME_DAY - **(C)**, 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 - **(C)**, 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 - **(C)**, название текущей публикации, заданное в органайзере (не название файла публикации).
PUBLICATION_PAGE_TITLE - **(C)**, 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_# - **(C)**, значение параметра введённого в командной строке. Отдельно создается для каждого параметра (**#** - номер параметра).
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 - позволяет ограничить доступ к главе на основе пароля. Используется только в том случае, если установлены пароли для глав на вкладке **"Password"** диалогового окна **"Publication Properties"**.

Синтаксис объектов

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 - оператор, создающий 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 операторы). 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 (_), например **Vector_1**, иначе объект не будет распознан скриптом. Лучше всего сразу переименовывать объекты в органайзере, убирая пробелы.

(**.**) - полная остановка (**full stop**), 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()
var BoolF=new Boolean(0)
var BoolF=new Boolean(null)
var BoolF=new Boolean("")
var BoolF=new Boolean(false)
```

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

```
var BoolT=new Boolean(true)
var BoolT=new Boolean(1)
var BoolT=new Boolean("true")
var BoolT=new Boolean("false")
var BoolT=new Boolean("any string")
```

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() - создание 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 и иначе)

Оператор **if** выполняет набор операторов, если проверяемое выражение (может состоять из нескольких выражений) истинно. Оператор **else** может быть добавлен к оператору **if**, и выполняет альтернативный набор операторов, если проверяемое выражение ложно. Оператор **if** может быть вложен в другой оператор **if** или **else**.

Синтаксис:

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

```
if (expression) {list of statements for true}
```

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

```
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 операторов (фрагмент кода), выполняемый если выражение истинно.

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

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

{} - набор операторов 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 оператор 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; var B=2; var 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 условие
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 содержащий условия
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

Функция **switch case** позволяет выполнять различные варианты наборов операторов в зависимости от значения проверяемой переменной. Если ни один конкретный вариант (**case**) не соответствует значению переменной, то выполняется набор операторов особого варианта, называемого **default**. В отличие от **C++** **case** может быть не только значением, но и выражением. Важно запомнить, что функция "проваливается", т.е. после выполнения списка операторов для конкретного варианта, функция перейдет к следующему варианту и выполнит также и его операторы. 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 операторов, то команда **break** после него обязательна, иначе выведется сообщение об ошибке. Команда **break** завершает функцию **switch case** после выполнения операторов варианта, соответствующего значению проверяемой переменной и не завершает функцию если вариант не соответствует значению 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) //проверяется переменная
{//начало функции
case n: //вариант, выполняющийся если значение переменной равно n
list of actions //набор операторов 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 actions //набор операторов 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 **9** (вместо **n**). Если число четное, то в скриптовой консоли на выводе будет фрагмент или полный текст о волке, если нечетное, то о зайце, и наконец если значение **x** иное (например **0**,**-1**, **10**,**"лошадь"**), то на выводе будет слово **"КОНЕЦ"**. Чем меньше число, тем полнее будет текст и меньше заглушек в виде **"*"**.

```
var x=n //вместо n надо подставить число от 1 до 9, или что-то иное для варианта default
var y=10 //эта переменная введена для демонстрации использование выражений в вариантах
var a="*";var b="*";var c="*";var d="*";var 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-8: a="Заяц"
case y-6: b=" весело"
case y-4: c=" играет"
case y-2: d=" на";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 выполняющийся 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 (**true**), и прекращает выполнение данного фрагмента кода, если проверяемое выражение цикла становится ложным (**false**). 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**. Специальный цикл **for in** позволяет 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"
Tapirica(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 ноль, 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)}
```

Синтаксис математических функций

Математические объекты **Math** и **Number** позволяют производить 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, данное 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).

Функции:

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.NaN получить не число **"Not-a-Number"**.
Number.NEGATIVE_INFINITY получить отрицательную бесконечность.
Number.POSITIVE_INFINITY получить положительную бесконечность.

Math.abs

Позволяет получить модуль числа.

Синтаксис:

Math.abs(x)

Возврат:

(Ч), абсолютное (положительное) исходное число.

Параметры:

x - **(Ч)**, исходное. **ОП**.

Пример:

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

Math.round

Позволяет округлить число до ближайшего целого.

Синтаксис:

Math.round(x)

Возврат:

(Ц), округлённое в ближайшую сторону исходное число.

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

Позволяет округлить число до большего целого.

Синтаксис:

Math.ceil(x)

Возврат:

(Ц), округлённое в большую сторону исходное число.

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

Позволяет округлить число до меньшего целого.

Синтаксис:

Math.floor(x)

Возврат:

(Ц), округлённое в меньшую сторону исходное число.

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

Позволяет 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

Позволяет получить максимум.

Синтаксис:

Math.max(x,y)

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

Параметры:

x - **(Ч)**, исходное. **ОП**.

y - **(Ч)**, исходное. **ОП**.

Пример:

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

Math.min

Позволяет получить минимум.

Синтаксис:

Math.min(x,y)

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

Параметры:

x - **(Ч)**, исходное. **ОП**.

y - **(Ч)**, исходное. **ОП**.

Пример:

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

Math.pow

Позволяет возвести число в степень.

Синтаксис:

Math.pow(x,y)

Возврат:

(Ч), результат возведения числа 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

Позволяет получить экспоненту числа.

Синтаксис:

Math.exp(x)

Возврат:

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

Параметры:

x - **(Ч)**, исходное. **ОП**.

Пример:

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

Math.atan2

Позволяет 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

Позволяет 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

Позволяет получить арксинус числа.
Синтаксис:
Math.asin(x)
Возврат:
(Ч), арксинус числа (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

Позволяет 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

Позволяет получить арккосинус числа.
Синтаксис:
Math.acos(x)
Возврат:
(Ч), арккосинус числа (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

Позволяет 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

Позволяет получить арктангенс числа.

Синтаксис:

```
Math.atan(x)
```

Возврат:

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

Параметры:

x - **(Ч)**, исходное. **ОП**.

Пример:

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

Math.log

Позволяет получить натуральный логарифм числа.

Синтаксис:

```
Math.log(x)
```

Возврат:

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

Параметры:

x - **(Ч)**, исходное. **ОП**.

Пример:

```
//Логарифм 8 по основанию 2
Debug.trace("LOG2 8="+Math.log(8)/Math.log(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 radius=8 //радиус круга
Debug.trace("Площадь круга с радиусом "+radius+": "+Math.round(Math.PI*Math.pow(radius,2)))
```

Math.sqrt

Позволяет получить квадратный корень числа.
Синтаксис:
Math.sqrt(x)
Возврат:
(Ч), квадратный корень исходного числа.
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.NaN

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

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

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)
```

Синтаксис функций, связанных со временем

Объект **Date** и связанные с ним функции позволяют производить различные операции с датой и временем, и взаимодействовать 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 таймера **Clock** и связанные с ним функции позволяют создавать таймеры 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(parameters)

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 - оператор, создающий 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.
parameters - значение даты и времени, может быть введено как числовое значение (количество миллисекунд, прошедших с полуночи **01.01.1970**), либо как строковое значение (по следующему образцу **"13 Month 1990, 00:00:00"**). **НП**, по умолчанию назначаются текущие системная дата и время компьютера. 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.

Функции:

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 временем.
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 таймер.

getFullYear и getUTCFullYear

Позволяют получить от указанного объекта даты и времени текущий год.

Синтаксис:

getFullYear()
getUTCFullYear()

Возврат:

(Ц), текущий год (четырехзначный).

Пример:

```
var Date1=new Date() //создание нового объекта даты
Debug.trace("Текущий год: "+Date1.getFullYear()) //вывод результата в скриптовую консоль
```

setFullYear и setUTCFullYear

Позволяют 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) //создание нового объекта даты и изменение в ней года
Debug.trace("Новый год: "+Date1.getFullYear()) //вывод результата в скриптовую консоль
```

getMonth и getUTCMonth

Позволяют 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() //получение месяца в новом объекте даты
Debug.trace("Текущий месяц: "+Month1+1) //вывод результата в скриптовую консоль
```

setMonth и setUTCMonth

Позволяют 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() //создание нового объекта даты
Date1.setMonth(5) //изменение месяца на июнь
Debug.trace("Месяц изменен: "+Date1) //вывод результата в скриптовую консоль
```

getDay и getUTCDay

Позволяют 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("День недели: "+Date1.getDay()) //вывод текущего дня недели
```

getDate и getUTCDate

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

Синтаксис:

getDate()
getUTCDate()

Возврат:

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

Пример:

```
var Date1=new Date();var MDay1=Date1.getDate() //получение числа в новом объекта даты
Debug.trace("Текущее число: "+MDay1) //вывод результата в скриптовую консоль
```

setDate и setUTCDate

Позволяют 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 ноль. **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) //создание нового объекта даты и изменение числа на 13-ое
Debug.trace("Число изменено: "+Date1) //вывод результата в скриптовую консоль
```

getHours и getUTCHours

Позволяют получить от указанного объекта даты и времени текущий час.

Синтаксис:

getHours()
getUTCHours()

Возврат:

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

Пример:

```
var Date1=new Date();var Hour1=Date1.getHours() //создание нового объекта даты и получение часа
Debug.trace("Текущий час: "+Hour1) //вывод результата в скриптовую консоль
```

setHours и setUTCHours

Позволяют 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) //создание нового объекта даты и изменение часа на 14
Debug.trace("Час изменен: "+Date1) //вывод результата в скриптовую консоль
```

getMinutes и getUTCMinutes

Позволяют получить от указанного объекта даты и времени текущее количество минут.

Синтаксис:

getMinutes()
getUTCMinutes()

Возврат:

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

Пример:

```
var Date1=new Date();var Min1=Date1.getMinutes() //создание нового объекта даты и получение минут
Debug.trace("Сколько минут? "+Min1) //вывод результата в скриптовую консоль
```

setMinutes и setUTCMinutes

Позволяют 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 **1** минута следующего часа). **ОП**.

Пример:

```
var Date1=new Date();Date1.setMinutes(39) //создание нового объекта даты и изменение минут
Debug.trace("Минуты изменены: "+Date1) //вывод результата в скриптовую консоль
```

getSeconds и getUTCSeconds

Позволяют получить от указанного объекта даты и времени текущее количество секунд.

Синтаксис:

getSeconds()
getUTCSeconds()

Возврат:

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

Пример:

```
var Date1=new Date();var Sec1=Date1.getSeconds() //создание нового объекта даты и получение секунд
Debug.trace("Сколько секунд? "+Sec1) //вывод результата в скриптовую консоль
```

setSeconds и setUTCSeconds

Позволяют 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 **1** секунда следующей минуты). **ОП**.

Пример:

```
var Date1=new Date();Date1.setSeconds(48) //создание нового объекта даты и изменение секунд
Debug.trace("Секунды изменены: "+Date1) //вывод результата в скриптовую консоль
```

getMilliseconds и getUTCMilliseconds

Позволяют получить от указанного объекта даты и времени текущее количество миллисекунд.

Синтаксис:

getMilliseconds()
getUTCMilliseconds()

Возврат:

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

Пример:

```
var Date1=new Date() //создание нового объекта даты
var MSec1=Date1.getMilliseconds() //получение миллисекунд
Debug.trace("Сколько миллисекунд? "+MSec1) //вывод результата в скриптовую консоль
```

setMilliseconds и setUTCMilliseconds

Позволяют 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 **1** миллисекунда следующей секунды). **ОП**.

Пример:

```
var Date1=new Date() //создание нового объекта даты
Date1.setMilliseconds(256) //изменение миллисекунд
Debug.trace("Миллисекунды изменены: "+Date1.getMilliseconds()) //вывод результата в консоль
```

getTime

Позволяет 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() //создание нового объекта даты
Date1.setMilliseconds(256) //изменение миллисекунд
var Date2=new Date() //создание второго объекта даты со значением текущего времени
Debug.trace("Разница: "+(Date2.getTime()-Date1.getTime())) //вывод результата в консоль
```

setTime

Позволяет 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() //создание нового объекта даты и времени
Date1.setTime(Date.parse("3 Sep 2006, 19:08:43")) //установка новой даты и времени
Debug.trace("Новые дата и время: "+Date1) //вывод результата в скриптовую консоль
```

Date.parse

Позволяет 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() //создание нового объекта даты и времени
var FromString=Date.parse("3 Sep 2006, 19:08:43") //перевод текста в миллисекунды
Date1.setTime(FromString) //установка переведенных миллисекунд в дату и время
Debug.trace("Время, конвертированное из текста: "+Date1) //вывод результата в консоль
```

getTimezoneOffset

Позволяет 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() //создание нового объекта даты и времени
Debug.trace("Разница в минутах: "+Date1.getTimezoneOffset()) //вывод результата в консоль
```

toLocaleString

Позволяет 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() //создание нового объекта даты и времени
Debug.trace("Время местного часового пояса: "+Date1.toLocaleString()) //вывод результата в консоль
```

toUTCString

Позволяет преобразовать в строку, переведенный в универсальное время, объект даты и времени.

Синтаксис:

toUTCString()

Возврат:

(С), универсальное время.

Пример:

```
var Date1=new Date() //создание нового объекта даты и времени
Debug.trace("Универсальное время: "+Date1.toUTCString()) //вывод результата в консоль
```

toGMTString

Позволяет 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() //создание нового объекта даты и времени
Debug.trace("Среднее время по Гринвичу: "+Date1.toGMTString()) //вывод результата в консоль
```

CreateClock

Позволяет создать объект таймера **Clock**. Таймер измеряет часы, минуты и секунды, и может отображаться на странице внутри названной переменной. Созданный таймер не запустится без функции **Start**.

Синтаксис:

CreateClock(Name,Variable,Format)

Параметры:

Name - **(С)**, название объекта таймера. **ОП**.

Variable - **(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]"**.

Start

Позволяет запустить созданный объект таймера и возобновить таймер с нуля после остановки или паузы.

Синтаксис: **Start()**

Stop

Позволяет остановить объект таймера в его текущем времени.

Синтаксис: **Stop()**

Pause

Позволяет поставить таймер на паузу в его текущем времени.

Синтаксис: **Pause()**

Continue

Позволяет снять таймер с паузы.

Синтаксис: **Continue()**

GetClock

Позволяет получить объект созданного таймера.

Синтаксис:

GetClock(Name)

Возврат: **(О)**, таймер.

Параметры:

Name - **(С)**, название объекта таймера. **ОП**.

GetHours

Позволяет получить количество часов, в течение которых воспроизводился объект таймера.

Синтаксис:

GetHours()

Возврат: **(Ц)**, количество часов, прошедших со старта таймера.

GetMinutes

Позволяет получить количество минут, в течение которых воспроизводился объект таймера.

Синтаксис:

GetMinutes()

Возврат: **(Ц)**, количество минут, прошедших со старта таймера.

GetSeconds

Позволяет получить количество секунд, в течение которых воспроизводился объект таймера.

Синтаксис:

GetSeconds()

Возврат: **(Ц)**, количество секунд, прошедших со старта таймера.

Общий пример:

```
var Timer1=0;var Timer1VAR=0;var Timer1Format="[hh] ч [mm] мин [ss] сек"
var Clock1=CreateClock("Timer1","Timer1VAR",Timer1Format)
Clock1.Start();Clock1.Stop();Clock1.Start();Clock1.Pause();Clock1.Continue();wait(3)
Debug.trace(Clock1.GetHours()+"ч "+Clock1.GetMinutes()+"м "+Clock1.GetSeconds()+"с")
var Clock1=GetClock("Timer1");Text1.SetSelection(0,-1);Text1.ReplaceSelection(Timer1VAR)
```

Строковые объекты

1) Глобальный строковый объект **String** позволяет производить операции с различными типами данных, работая 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 - оператор объявления новой переменной.
new - оператор 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).

Функции 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.
Глобальные строковые функции:
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.

Функции 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

Позволяет 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") //в скриптовой консоли выведется 18
Debug.trace(String.length(9999)+"\n") //в скриптовой консоли выведется 4
Debug.trace(String.length(Vector1)) //выведется 19, длина названия объекта "[ILMObject Vector1]"
```

String.mid

Позволяет 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)) //в скриптовой консоли выведется слово "Новый"
```

String.left

Позволяет 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 целиком всё исходное строковое значение). **ОП**.
Пример:
```
Debug.trace(String.left("Волкодав",4)) //в скриптовой консоли выведется слово "Волк"
```

String.right

Позволяет 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 целиком всё исходное строковое значение). **ОП**.

Пример:

```
Debug.trace(String.right("Трубкозуб",3)) //в скриптовой консоли выведется слово "зуб"
```

String.toupper

Позволяет 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") //в скриптовой консоли выведется "MINOTAURUS"
Debug.trace(String.toupper(Vector1)) //в скриптовой консоли выведется "[ILMOBJECT VECTOR1]"
```

String.tolower

Позволяет 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") //в скриптовой консоли выведется "minotaurus"
Debug.trace(String.tolower(Vector1)) //в скриптовой консоли выведется "[ilmobject vector1]"
```

String.contains

Позволяет 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

```
var Phrase1="Labyrinth of the Minotaur";var Searchword1="mInOtAuR"
Debug.trace(String.contains(Phrase1,Searchword1)) //в скриптовой консоли выведется false
```

С игнорированием регистра

```
var Phrase1="Labyrinth of the Minotaur";var Searchword1="mInOtAuR"
Debug.trace(String.contains(Phrase1,Searchword1,true)) //в скриптовой консоли выведется true
```

С русскими буквами

```
Debug.trace(String.contains("Я ломал стекло как шоколад в руке","шоколад")) //на выводе будет true
Debug.trace(String.contains("Я ломал стекло как шоколад в руке","мармелад")) //на выводе будет false
```

String.word

Позволяет 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:

```
var LongText="Жираф#@Носорог#@Бегемот#@Улитка#@Пингвин"
Debug.trace(String.word(LongText,4,"#@")) //в скриптовой консоли на выводе будет слово "Улитка"
```

String.fromCharCode

Позволяет 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

Позволяет 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))) //вывод случайного числа от 1 до 99
Debug.trace("\n"+String.random(2)) //вывод в скриптовую консоль либо 0, либо 1
Debug.trace("\n"+(40+String.integer(String.random(11)))) //вывод случайного числа от 40 до 50
```

String.format

Позволяет 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).

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

Примеры:

```
Debug.trace(String.format("09.3",1.1)) //в скриптовой консоли на выводе будет значение "00001.100"
```

String.bool

Позволяет 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"+"рука=рука="+String.bool("рука"=="рука")+"(булево)")
```

String.integer

Позволяет 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

Позволяет 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"+"рука=рука="+String.number("рука"=="рука")+"(числовое)")
```

String.string

Позволяет 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"+"рука=река="+String.string("рука"=="река")+"(строковое)")
```

escape и unescape

escape позволяет закодировать в строковом значении специальные символы за исключением: + - * / @ _

unescape позволяет вернуть исходное состояние строкового значения с закодированными спецсимволами.

Синтаксис:

escape(String)

unescape(String)

Возврат:

(С)

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

Позволяет 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:
Подсчет количества строчек в строковом значении:

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

Подсчет количества строчек в строковом объекте:

```
var Phrase2=new String("Черепаха\nЯщерица\nДинозавр")
Debug.trace(CountLines(String.string(Text2)))
Debug.trace("\n"+CountLines(Phrase2.toString()))
```

Подсчет количества строчек в текстовом файле:

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

Подсчет количества строчек в текстовом объекте:

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

CountWords

Позволяет 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:
Подсчет количества слов в строковом значении:

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

Подсчет количества слов в строковом объекте:

```
var Phrase2=new String("Кто бежит быстрее по тропинкам сада, тот получит первым приз из шоколада")
Debug.trace(CountWords(String.string(Text2)))
Debug.trace("\n"+CountWords(Phrase2.toString()))
```

Подсчет количества слов в текстовом файле:

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

Подсчет количества слов в текстовом объекте:

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

StringReplace

Позволяет 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)) //в скриптовой консоли выведется "[ILMObject Text8]"
```

ToClipboard

Позволяет 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

Позволяет распечатать указанное строковое значение со шрифтом поумолчанию.

Синтаксис:

PrintString(String,PrintDialog,CancelDialog,Landscape)

Параметры:

String - **(С)**, для печати.

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

CancelDialog - **(Б)**, отображение диалогового 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

Позволяет 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

Позволяет 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) //в скриптовой консоли выведется 20
```

charAt

Позволяет 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)) //в скриптовой консоли выведется "м" третий символ строки
```

charCodeAt

Позволяет 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)) //в скриптовой консоли выведется число 38 - код символа "&"
```

toUpperCase

Позволяет 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()) //в скриптовой консоли выведется "MINOTAURUS"
```

toLowerCase

Позволяет 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 верхнем регистре.
Пример:

```
var Word1="MiNoTaUrUs"
Debug.trace(Word1.toLowerCase()) //в скриптовой консоли выведется "minotaurus"
```

substring

Позволяет 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)) //в скриптовой консоли выведется слово "минотавр"
```

split

Позволяет 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(1);Debug.trace(Array1[m]+"\n")} //вывод значений пяти элементов массива
```

indexOf и lastIndexOf

Позволяют 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 возвращается значение **-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 позволяют открывать, читать и изменять файлы, используя скрипт. Чтобы использовать эти функции, необходимо сначала использовать функцию **OpenFile** для создания нового файлового объекта в скрипте. 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"**).

Функции:

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

Позволяет 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 в консоль если файла нет
else {Debug.trace("Файл существует")} //вывод результата в консоль если файл существует
```

GetFileVersion

Позволяет 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

Позволяет 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:

```
//Файл откроется в связанной с ним программе
LaunchFile(SYSTEM_PUBLICATION_DIR+"\\Readme.txt")
//Запустится программа, а в ней откроется текстовый файл из параметров
var FileName=SYSTEM_WIN_DIR+"\\notepad.exe"
var Params=SYSTEM_PUBLICATION_DIR+"\\Readme.txt"
LaunchFile(FileName,Params)
```

CopyFile

Позволяет 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 диалогового 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

Позволяет 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 диалогового 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

Позволяет распечатать указанный файл.

Синтаксис:

PrintFile(Filename,PrintDialog,CancelDialog)

Параметры:

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

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

CancelDialog - **(Б)**, отображение диалогового 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

Позволяет 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

Позволяет читать файл, открытый/созданный функцией **OpenFile**.

Синтаксис:

Read(Encrypt,Key)

Возврат:

(С), весь текст файла.

Параметры:

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

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

ReadLine

Позволяет 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. **НП**.

ReadField

Позволяет 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. **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

Позволяет 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

Позволяет записать строковое значение **на новой строчке в конце файла**,открытого/созданного функцией **OpenFile**, если эта функция используется после такой же, а не после функций **Write** и **Writefield**.

Синтаксис:

WriteLine(String,Quote,Encrypt,Key)

Параметры:

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

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

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

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

WriteField

Позволяет записать **через запятую** строковое значение **в конец последней существующей строчки файла**, открытого/созданного функцией **OpenFile**, если эта функция используется после такой же, а не после функций **Write** и **Writeline**.

Синтаксис:

WriteField(String,Quote,Encrypt,Key)

Параметры:

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

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

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

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

Close

Позволяет 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("Поле носорогов") //запись в файл
Debug.trace(Data1.Read()) //чтение всего текста
Data1.Close() //закрытие файла
```

GotoFirstLine

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

Синтаксис:

GotoFirstLine()

GotoNextLine

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

Синтаксис:

GotoNextLine()

EndOfFile

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

Синтаксис:

EndOfFile()

Возврат:

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

Общий пример:

```
var Text1=OpenFile(SYSTEM_PUBLICATION_DIR+"DATA\\TestText.txt") //открытие файла
Text1.GotoNextLine() //переход к следующей строчке файла
Text1.GotoNextLine() //переход к следующей строчке файла
Text1.GotoFirstLine() //переход к первой строчке файла
Debug.trace("Чтение только первой строки: "+Text1.ReadLine()) //чтение первой строчки
Text1.GotoFirstLine() //переход к первой строчке файла
while (Text1.EndOfFile()==false) //чтение текста из файла с начала, пока он не кончится
{wait(1);Debug.trace("\n"+"Чтение пока не кончится текст:"+Text1.ReadLine())}
Debug.trace("\n"+"КОНЕЦ") //выводится, когда все строки файла прочитаны
```

HtmlHelp и WinHelp

Позволяют открывать файлы справок (**CHM**, **HLP**) в указанном разделе (для поддержки **CHM** требуется версия **Windows** старше **2000**, с **Internet Explorer 4** или выше).

Синтаксис:

HtmlHelp(Filename,ID)

WinHelp(Filename,ID)

Параметры:

Filename - **(С)**, путь к 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)
```

GetINIFileData и GetINISectionData

Позволяют получить значения из указанного **INI**-файла.

Синтаксис:

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

Возврат:

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

Параметры:

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

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

Позволяет 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

Позволяет 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

Позволяет 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

Позволяет 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)
```

Базовые объекты

Функции базовых объектов используются со всеми объектами, отображающимися 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

Функции:

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 рамочный объект с названием **Frame1**. Внутри рамочного объекта надо создать два простых векторных объекта с названиями **Vector1** и **Vector2**, содержащими внутри по одному полигону с названиями **Polygon1** и **Polygon2** соответственно.

GetName

Позволяет получить название объекта, заданное в органайзере

Синтаксис:

GetName()

Возврат:

(C), название указанного объекта.

Пример:

```
Debug.trace("Название объекта: "+this.GetName()) //вывод результата в скриптовой консоли
```

GetType

Позволяет получить тип указанного 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()) //вывод результата в скриптовой консоли
```

GetPage

Позволяет 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("Находится на странице: "+Frame1.GetPage()) //вывод результата в скриптовой консоли
```

GetUniqueObjectID

Позволяет получить уникальный идентификатор указанного объекта.

Синтаксис:

GetUniqueObjectID()

Возврат:

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

Пример:

```
Debug.trace("Уникальный идентификатор: "+this.GetUniqueObjectID()) //вывод результата в скриптовой консоли
```

GetResource

Позволяет получить информацию о ресурсе указанного 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

Позволяет получить конкретный дочерний элемент указанного объекта.

Синтаксис:

FindChild(Name)

Возврат:

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

Параметры:

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

Примеры:

```
var VChild1=Vector1.FindChild("Polygon1") //нахождение заданного дочернего элемента в объекте
Debug.trace(VChild1) //вывод результата в скриптовой консоли
```

FindDescendant

Позволяет получить конкретного потомка указанного 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:

```
var FDesc1=Frame1.FindDescendant("Polygon2") //нахождение  заданного потомка в объекте
Debug.trace(FDesc1) //вывод результата в скриптовой консоли
```

GetNumberChildren

Позволяет получить количество дочерних элементов указанного объекта.

Синтаксис:

GetNumberChildren()

Возврат:

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

Пример:

```
var ChildNumb=Frame1.GetNumberChildren() //нахождение количества дочерних элементов объекта
Debug.trace(ChildNumb) //вывод результата в скриптовой консоли
```

GetParent

Позволяет получить родительский элемент указанного объекта.

Синтаксис:

GetParent()

Возврат:

(О), родительский элемент 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:

```
var V1Par=Vector1.GetParent() //переменной присваивается родительский элемент заданного объекта
Debug.trace(V1Par) //вывод результата в скриптовой консоли
```

GetChild

Позволяет получить по номеру дочерний элемент указанного объекта.

Синтаксис:

GetChild(Index)

Возврат:

(О), указанный дочерний элемент объекта.

Параметры:

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

Пример:

```
var Ch2=Frame1.GetChild(1);Debug.trace(Ch2)
```

GetFirstChild

Позволяет получить первый дочерний элемент указанного объекта.

Синтаксис:

GetFirstChild()

Возврат:

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

Пример:

```
var Ch1=Frame1.GetFirstChild();Debug.trace(Ch1)
```

GetNextChild

Позволяет получить следующий дочерний элемент указанного 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:

```
var Ch2=Frame1.GetNextChild(Vector1);Debug.trace(Ch2)
```

Публикации

Публикация и страница - это глобальные объекты, а функции - это свойства этих объектов.

Функции:

ChangeDisplayMode изменить разрешение экрана во время работы публикации.
RGB получить значение цвета из количеств красного, зеленого и синего.
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 для системного реестра.
GetLocale получить текущий язык операционной системы.

Главы

Функции главы позволяют управлять окнами в публикации.

Функции:

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

ChangeDisplayMode

Позволяет изменять 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, содержащий возможные разрешения экрана. Возможные значения: **0** - **640x480**, **1** - **800x600**, **2** - **1024x768** и т.д. **НП**, по умолчанию **-1** (разрешение экрана устройства, как оно было до запуска публикации).
DepthIndex - **(Ц)**, глубина цвета на экране. Возможные значения: **15**, **16**, **24**, **36**. **НП**, по умолчанию **-1** (глубина цвета на экране устройства, как она была до запуска публикации).

Пример:

```
ChangeDisplayMode(1) //изменение разрешения экрана на 800x600
```

RGB

Позволяет задать значение цвета, которое можно применить к объектам с помощью функций изменения цвета.

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 Lava=52;var Grass=13;var Sky=113
var Color1=RGB(Lava,Grass,Sky) //фиолетовый цвет
var Color2=RGB(130,202,46) //травяной цвет
```

GetPublication

Позволяет получить текущую публикацию как объект.

Синтаксис:

GetPublication()

Возврат:

(О), текущая публикация.

Пример:

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

ResetVariables

Позволяет 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

Позволяют сохранить и 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()
RestorePublicationState()
```

IsPreview

Позволяет 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

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

Пояснение:

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

Позволяет 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

Позволяет 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

Позволяет 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("Rakapom"))
```

UpdateEvalState

Позволяет обновить текущее состояние пробного периода публикации. Полезна 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. Он просто обновляет системную переменную **PUBLICATION_EVALUATION** с правильной информацией.

Синтаксис:

UpdateEvalState()

Пример:

```
UpdateEvalState()
```

ReadRegistryKey

Позволяет получить значение определенного раздела 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

Позволяет записать значение. 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

Позволяет удалить значение. 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

Позволяет получить язык операционной системы (НЕ язык ввода с клавиатуры).

Синтаксис:

GetLocale(Info)

Параметры:

Info - **(Ц)**, тип информации, возможные значения: **0**-трехбуквенный код страны, **1**-двухбуквенный код страны, **2**-двухбуквенный код языка. **НП**, по умолчанию **0**.

Возврат:

(С), язык операционной системы.

Пример:

```
Debug.trace("Язык системы: "+GetLocale(0))
```

SetInitialPositionExact и SetInitialPositionPercent

Позволяют установить предварительное расположение окна на экране, но не изменить расположение уже открытого окна.

Синтаксис:

SetInitialPositionExact(PosX,PosY)
SetInitialPositionPercent(PercentageX,PercentageY)

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**.
PercentageX - **(Ц)**, процентное расположение окна на экране по горизонтали. **-1** для исходного состояния. **ОП**.
PercentageY - **(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) //установка положения окна
GotoPage(Chap.GetFirstChild().GetName()) //открыть главу, перейдя на первую страницу в ней
```

GetWindowState

Позволяет получить свойства указанного окна.

Синтаксис:

GetWindowState()

Возврат:

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

GetView

Позволяет получить текущее окно.

Синтаксис:

GetView()

Возврат:

(О), текущее окно.

Пример:

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

GetViewAt

Позволяет 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

Позволяет получить первое из открытых на данный момент окон.

Синтаксис:

GetFirstView()

Возврат:

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

Пример:

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

GetNextView

Позволяет 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()
while (TestView) {Debug.trace(TestView.GetPage().GetName());TestView=GetNextView(TestView)}
```

GetNumberViews

Позволяет получить количество открытых на данный момент окон.

Синтаксис:

GetNumberViews()

Возврат:

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

Пример:

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

GetType

Позволяет получить тип указанного окна.

Синтаксис:

GetType()

Возврат:

(Ц), тип 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())
```

Close

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

Синтаксис:

Close()

Пример:

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

Страницы

Функции:
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.
GetPageDownloadPercent получить процент завершения загрузки страницы.
GetPageDownloadPosition получить позицию загруженной страницы.
GetPageDownloadTotal получить общий размер загруженной страницы.
CopyToClipboard скопировать страницу в буфер обмена **Windows**.
PrintPage распечатать страницу.
ShowBookmarkDialog показать диалоговое 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

Позволяет 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**. Объявления переменных страницы: пункт меню **"Edit"** > команда **"Page Properties" >** вкладка **"Variables"**.

Синтаксис:
ResetVars()
Пример:
ResetVars()

GotoRandomPage

Позволяет 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

Позволяет 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

Позволяет перейти на указанную страницу любой главы.

Синтаксис:

GotoPage(Page)

Параметры:

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

Примеры:

GotoPage("Page 1")

GotoForwardPage

Позволяет перейти на следующую страницу текущей главы.

Синтаксис:

GotoForwardPage()

Пример:

GotoForwardPage()

GotoBackwardPage

Позволяет перейти на предыдущую страницу текущей главы.

Синтаксис:

GotoBackwardPage()

Пример:

GotoBackwardPage()

GotoNextPage

Позволяет 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

Позволяет 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

Позволяет 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

Позволяет 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(GetPageNumber()) //вывод результата в скриптовую консоль
```

GetPage

Позволяет получить указанную 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**.

Примеры:

```
//В свежесозданной публикации страница с названием "Page 1" - это страница под номером 0
Debug.trace(GetPage("Page 1")+"\n"+GetPage(0)) //вывод результата в скриптовую консоль
```

GetLastPage

Позволяет получить последнюю страницу для использования с другими функциями.

Синтаксис:

GetLastPage()

Возврат:

(О), последняя страница (всей публикации).

Примеры:

```
Debug.trace(GetLastPage()) //вывод результата в скриптовую консоль
```

GetNextPage

Позволяет 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()) //вывод результата в скриптовую консоль
```

GetPreviousPage

Позволяет 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()) //вывод результата в скриптовую консоль
```

GetPageDownloadPosition

Позволяет получить позицию загружаемой страницы.

Синтаксис:

GetPageDownloadPosition(Page)

Возврат:

(Ц), позиция загруженной страницы.

Параметры:

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

Пример:

GetPageDownloadPosition("Page1")

GetPageDownloadTotal

Позволяет получить общее количество загруженной страницы.

Синтаксис:

GetPageDownloadTotal(Page)

Возврат:

(Ц), общий объем загруженной страницы.

Параметры:

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

Пример:

GetPageDownloadTotal("Page1")

GetPageDownloadPercent

Позволяет получить процент указанной страницы, которая была загружена в данный момент.

Синтаксис:

GetPageDownloadPercent(Page)

Возврат:

(Ц), процент загруженной страницы.

Параметры:

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

Пример:

GetPageDownloadPercent("Page1")

CopyToClipboard

Позволяет скопировать изображение текущей страницы в буфер обмена **Windows**.

Синтаксис:

CopyToClipboard()

Пример:

CopyToClipboard()

PrintPage

Позволяет распечатать указанную 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 диалогового окна свойств печати перед распечаткой (**true** - окно появится, **false** - нет). **НП,** по умолчанию **false**.

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

Пример:

PrintPage("Page1", true, false)

ShowBookmarkDialog

Позволяет вывести на экран встроенное диалоговое окно закладок **"Bookmarks"**.

Синтаксис:

ShowBookmarkDialog()

Пример:

ShowBookmarkDialog()

SetBookmark

Позволяет назначить закладку для текущей страницы.

Синтаксис:

SetBookmark()

Пример:

SetBookmark()

ClearBookmark

Позволяет убрать закладку с текущей страницы.

Синтаксис:

ClearBookmark()

Пример:

ClearBookmark()

GotoBookmark

Позволяет перейти к ближайшей странице с закладкой.

Синтаксис:

GotoBookmark()

Пример:

GotoBookmark()

GetBookmarkPage

Позволяет 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

Позволяет получить первую страницу с закладкой.

Синтаксис:

GetFirstBookmark()

Возврат:

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

Пример:

GetFirstBookmark()

GetNextBookmark

Позволяет получить следующую страницу с закладкой.

Синтаксис:

GetNextBookmark()

Возврат:

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

Пример:

GetNextBookmark()

Графические объекты

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

Функции:
Enable включить или отключить объект.
IsEnabled проверить, включен ли объект.
Show показать объект.
Hide скрыть объект.
IsShowing проверить, показывается ли объект.
GetLayer получить слой, на котором находится объект.
SetLayer установить слой, на котором находится объект.
CloneObject клонировать объект.
DestroyClonedObject удалить клонированный объект.
IsObjectIntersecting проверить, пересекается ли объект с другим.
GetWidth получить ширину объекта.
GetHeight получить высоту объекта.
SetObjectSize установить ширину и высоту объекта.
GetPersistentObject получить объект для хранения информации, доступной, когда страница не видна.
GetObjectDimensions получить положение и размер объекта.
SetObjectDimensions установить положение и размер объекта.
GetDisplayData получить свойства трансформации объекта.
SetDisplayData установить отображение информации, связанной с объектом.
GetPosition (для всех графических объектов) получить координаты положения объекта.
GetXPosition получить **X**-координату положения объекта.
GetYPosition получить **Y**-координату положения объекта.
GetPosition (для объекта 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 заставить объект следовать по анимационному пути.
SetPosition установить координаты положения объекта.
SetPositionX установить **X**-координату положения объекта.
SetPositionY установить **Y**-координату положения объекта.
Move переместить объект по его координатам.
MoveX переместить объект по его **X**-координатам.
MoveY переместить объект по его **Y**-координатам.
SetTransparency установить прозрачность объекта.
Fade сделать объект прозрачнее.
SetRotation установить поворот объекта вокруг центра.
Rotate повернуть объект вокруг центра.
SetRoll назначить поворот объекта вокруг **X**-оси.
Roll повернуть объект вокруг **X**-оси.
SetSpin назначить поворот объекта вокруг **Y**-оси.
Spin повернуть объект вокруг **Y**-оси.
SetScale установить горизонтальный и вертикальный масштаб объекта в процентах.
SetScaleH установить горизонтальный масштаб объекта в процентах.
SetScaleV установить вертикальный масштаб объекта в процентах.
Scale масштабировать объект по горизонтали и вертикали.
ScaleH масштабировать объект по горизонтали.
ScaleV масштабировать объект по вертикали.
SetSkew наклонить объект горизонтально и вертикально.
SetSkewH наклонить объект горизонтально.
SetSkewV наклонить объект вертикально.
Skew назначить перекос объекта по горизонтали и вертикали.
SkewH назначить перекос объекта горизонтально.
SkewV назначить перекос объекта вертикально.
ResetAnimation сбросить анимацию.
StopAnimation остановить анимацию.
GetScrollInfo получить значения состояния прокрутки.
SetScrollPosition установить значения состояния прокрутки.
GetAppearance получить информацию о внешнем виде для объекта состояния указанного объекта
SetImage создать фоновое растровое изображение для объекта.
RemoveImage убрать фоновое растровое изображение для объекта.
ReloadImage перезагрузить фоновое растровое изображение для объекта.
SetAlpha создать альфа-эффект для объекта.
RemoveAlpha убрать альфа-эффект для объекта.
SetBackground создать фон для объекта.
RemoveBackground убрать фон для объекта.
SetBorder создать рамку для объекта.
RemoveBorder убрать рамку для объекта.
SetFlare создать эффект свечения для объекта.
RemoveFlare убрать эффект свечения для объекта.
SetShadow создать эффект тени для объекта.
RemoveShadow убрать эффект тени для объекта.
SetTexture создать эффект текстуры для объекта.
RemoveTexture убрать эффект текстуры для объекта.
SetBtnColour создать эффект кнопки для объекта.
RemoveBtnColour убрать эффект кнопки для объекта.
SetFocus направить весь ввод с клавиатуры на объект.
ClearInputFocus снять с объекта ранее направленный ввод с клавиатуры.
CaptureMouse направить весь ввод мышью на объект.
ReleaseMouse снять с объекта ранее направленный ввод мышью.
CopyToClipboard скопировать объект в буфер обмена.
PrintObject распечатать объект.

Enable

Позволяет включить или отключить указанный графический объект.

Особенности функции:

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

2) Отключенный объект продолжает отображаться на странице и на него действуют все скриптовые функции.

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

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

Синтаксис:

Enable(Set)

Параметры:

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

Примеры:

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

IsEnabled

Позволяет проверить текущее состояние работы указанного графического объекта, включен ли он или отключен.

Синтаксис:

IsEnabled()

Возврат:

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

Пример:

```
Debug.trace(Button1.IsEnabled()) //в скриптовой консоли выведется состояние работы кнопки
```

Show

Позволяет показать указанный 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 объекта при появлении после скрытия (**true** - исходное (заданное в редакторе), **false** - на месте скрытия). **НП**, по умолчанию **true**.

Примеры:

```
Vector1.Move(100,100,2) //перемещение векторного изображения
wait(1);Vector1.Hide() //скрытие векторного изображения
wait(1);Vector1.Show(false) //появление векторного изображения на месте скрытия
```

Hide

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

Синтаксис:

Hide()

Пример:

```
Vector1.Hide() //скрытие векторного изображения
```

IsShowing

Позволяет проверить текущее состояние отображения указанного графического объекта на странице.

Синтаксис:

IsShowing()

Возврат:

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

Пример:

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

GetLayer

Позволяет определить на каком слое располагается указанный графический объект.

Синтаксис:

GetLayer()

Возврат:

(Ц), номер слоя, где расположен указанный графический объект.

Пример:

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

SetLayer

Позволяет поместить указанный графический объект на определенный слой (**0** - слой расположения по умолчанию). Графические объекты на слое с большим значением отображаются впереди, а с меньшим - позади.

ВАЖНО: у графических объектов внутри кадра собственная нумерация слоёв, относительно расположения в кадре (**0** - слой расположения по умолчанию).

Синтаксис:

SetLayer(Number)

Параметры:

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

Пример:

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

CloneObject

Позволяет создать идентичную копию (клон) указанного 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 (**ИСКЛЮЧЕНИЕ:** 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 графический объект из прикрепленного к нему самому скриптового объекта.

Синтаксис:

CloneObject(PosX,PosY,Visible)

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.

Возврат:

(О), новый клон графического объекта.

Пример:

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

DestroyClonedObject

Позволяет удалить указанный 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) //клонирование векторного изображения
wait(5);Clone1.life=0 //обнуление пользовательского свойства у клона через 5 секунд
//Скрипт, прикрепленный к объекту Vector1:
this.life=1 //всем клонам векторного изображения будет присвоено данное пользовательское свойство
while(this.life==1){this.Spin(360,1)} //пока пользовательское свойство не обнулилось, клон вертится
this.DestroyClonedObject() //уничтожение клона при обнулении пользовательского свойства
```

IsObjectIntersecting

Позволяет проверить, пересекаются ли на текущей странице два указанных 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 **magenta** - сиреневый) будет указан как прозрачная область.

3) Часто поворот или скос объекта (его ограничительной рамки) в векторном редакторе программы приводит к вылету, либо к **неправильной** работе функции и искаженному отображению самого объекта в запущенной публикации. Это один из крупных багов программы, поэтому рекомендуется не поворачивать и не перекашивать объекты средствами редактора, а производить эти операции с помощью скриптовых команд.

Синтаксис:

FirstObject.IsObjectIntersecting(SecondObject)

Возврат:

(Б), результат проверки на пересечение (**true** - объекты пересекаются, **false** - не пересекаются).

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

Позволяет получить ширину ограничительной рамки указанного графического объекта.

Синтаксис:

GetWidth()

Возврат:

(Ц), ширина указанного графического объекта (в пикселях).

Пример:

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

GetHeight

Позволяет получить высоту ограничительной рамки указанного графического объекта.

Синтаксис:

GetHeight()

Возврат:

(Ц), высота указанного графического объекта (в пикселях).

Пример:

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

SetObjectSize

Позволяет назначить ширину и высоту ограничительной рамки указанного графического объекта.

Особенности функции:

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

2) Изменение растровых изображений зависит от свойств графического 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 объекта (в пикселях). **ОП**.

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

Пример:

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

GetPersistentObject

Позволяет создать уникальную постоянную копию указанного объекта.

Синтаксис:

GetPersistentObject()

Возврат:

(О), уникальная постоянная копия указанного объекта.

Пример:

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

GetObjectDimensions

Позволяет получить данные о координатах (относительно верхнего левого угла страницы) и размере указанного графического объекта (его ограничительной рамки). Если данный графический 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 указанного объекта.

height - **(Ц)**, высота ограничительной рамки указанного объекта.

left - **(Ц)**, **X**-координата левой стороны ограничительной рамки указанного объекта.

right - **(Ц)**, **X**-координата правой стороны ограничительной рамки указанного объекта.

top - **(Ц)**, **Y**-координата верхней стороны ограничительной рамки указанного объекта.

bottom - **(Ц)**, **Y**-координата нижней стороны ограничительной рамки указанного объекта.

Примеры:

```
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

Позволяет установить размеры указанного объекта равными размерам другого объекта. Данная функция работает **неправильно**:

1) Если ширина и высота исходного объекта **не равны** ширине и высоте того объекта, из которого брались свойства, то визуально исходный объект будет только растянут из исходной точки своего верхнего левого угла, по размеру того объекта, из которого были взяты свойства. Если же проверить свойства измененного исходного объекта функцией **GetObjectDimensions**, то ширина и высота останутся прежними, а исходные координаты сместятся так, как если бы исходный объект стоял в центре измененного себя.

2) Если ширина и высота исходного объекта **равны** ширине и высоте того объекта, из которого брались свойства, то исходный объект изменит свои координаты правильно, и переместится на то же место, где стоит объект из которого брались свойства.

Синтаксис:

SetObjectDimensions(DimensionObject)

Параметры:

DimensionObject - **(О)** со всеми или несколькими свойствами (описаны у функции **GetObjectDimensions**) для применения к указанному графическому объекту. **ОП**.

Общий пример:

```
var V1DIM=Vector1.GetObjectDimensions()  //переменная содержит свойства графического объекта
Vector2.SetObjectDimensions(V1DIM) //свойства применяются к другому графическому объекту
```

GetDisplayData

Позволяет получить свойства трансформации указанного графического объекта.

Особенности функции:

1) Начальные координаты любого объекта, располагающегося на любой позиции считаются равными **(0,0)**, они будут точкой отсчета для объекта.

2) Если кадр, содержащий объект, изменил координаты на странице, поменял угол или был масштабирован, то значения координат, угла и масштаба 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 графического объекта к другому, необходимо предварительно поделить значения этих двух свойств на **10**.

Синтаксис:

GetDisplayData()

Возврат:

(О) со следующими свойствами:

x - **(Ц)**, **X**-координата центра указанного объекта, относительно его начальной позиции.

y - **(Ц)**, **Y**-координата центра указанного объекта, относительно его начальной позиции.

scalex - **(Ч)**, текущая величина масштабирования указанного объекта в **X**-направлении.

scaley - **(Ч)**, текущая величина масштабирования указанного объекта в **Y**-направлении.

angle - **(Ч)**, текущий угол поворота в градусах указанного объекта вокруг центра.

anglex - **(Ч)**, текущий угол поворота в градусах указанного объекта вокруг **X**-оси в 3D.

angley - **(Ч)**, текущий угол поворота в градусах указанного объекта вокруг **Y**-оси в 3D.

skewx - **(Ч)**, текущая величина перекоса указанного объекта в **X**-направлении.

skewy - **(Ч)**, текущая величина перекоса указанного объекта в **Y**-направлении.

transparency - **(Ц)**, текущий уровень прозрачности указанного объекта (**0** - прозрачность отсутствует, **100** - полная прозрачность).

SetDisplayData

Позволяет установить свойства указанному графическому объекту.

Особенности функции:

1) Начальные координаты любого объекта, располагающегося на любой позиции считаются равными **(0,0)**, они будут точкой отсчета для объекта.

2) Если кадр, содержащий объект, изменил координаты на странице, поменял угол или был масштабирован, то значения координат, угла и масштаба объекта, расположенного внутри, не изменятся.

Синтаксис:

SetDisplayData(PositionObject)

Параметры:

PositionObject - **(О)** со всеми или несколькими свойствами (описаны у функции **GetDisplayData**) для применения к указанному графическому объекту. **ОП**.

Общий пример:

```
var ObjInfo=new Object() //создание объекта для хранения свойств и присвоение значений свойствам
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
wait(3);Vector1.SetDisplayData(ObjInfo) //передача свойств векторному изображению
var ObjInfo=Vector1.GetDisplayData() //взятие свойств у одного векторного изображения
ObjInfo.skewx/=10;ObjInfo.skewy/=10 //исправление ошибочных значений
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) //передача свойств другому векторному изображению
```

GetPosition, GetXPosition и GetYPosition

Позволяют получить координаты указанного графического объекта. Отсчет координат при использовании этих функций **всегда** происходит относительно верхнего левого угла страницы, даже если объект находится внутри кадра. Следует внимательно использовать эти функции вместе с **SetPosition**, **SetPositionX**, **SetPositionY**, из-за различий отсчета координат для объекта внутри кадра.

Синтаксис:

GetPosition()
GetXPosition()
GetYPosition()

Возврат:

GetPosition возвращает **(О)** со следующими свойствами:
x - **(Ч)**, **X**-координата центра указанного объекта относительно верхнего левого угла страницы.
y - **(Ч)**, **Y**-координата центра указанного объекта относительно верхнего левого угла страницы.
GetXPosition возвращает **(Ч)**, **X**-координату центра указанного объекта относительно верхнего левого угла страницы.
GetYPosition возвращает **(Ч)**, **Y**-координату центра указанного 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") //вывод координат в скриптовую консоль
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) //вывод координат в скриптовую консоль
```

SetPosition, SetPositionX, SetPositionY и Move, MoveX, MoveY

Позволяют изменить координаты центра указанного объекта. Если объект располагается на странице внутри кадра, то координаты отсчитываются от верхнего левого угла кадра, а если объект располагается на странице и не принадлежит кадру, то координаты отсчитываются от верхнего левого угла страницы. Если кадр, содержащий объект, повернут на угол (или перекошен), то и оси координат внутри фрейма повернутся на тот же угол (или перекосятся), и если объект внутри этого кадра соответственно координатам переместится прямолинейно, то выглядеть перемещение 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 указанного объекта по **X**-оси координат. **SetPosition** и **SetPositionX** изменят текущую **X**-координату объекта, на ту, что указана в параметре, а **Move** и **MoveX** переместят объект по **X**-оси координат на столько пикселей от текущей позиции, сколько указано в параметре. **ОП**.
PosY - **(Ц)** (может быть отрицательным), величина перемещения центра указанного объекта по **Y**-оси координат. **SetPosition** и **SetPositionY** изменят текущую **Y**-координату объекта, на ту, что указана в параметре, а **Move** и **MoveY** переместят 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) //изображение переместится в координаты 100,100 за 3.5 секунды
Vector1.Move(100,100,3.5) //изображение переместится по диагонали вправо-вниз за 3,5 секунды
```

GetPosition и GetPositionFromPercent

Позволяют получить координаты заданной точки на векторной линии указанного объекта анимационного пути.

Синтаксис:

GetPosition(Length)

GetPositionFromPercent(Percent)

Возврат:

(О) со следующими свойствами:

x - **(Ц)**, **X**-координата заданной точки на векторной линии анимационного пути, относительно центра ограничительной рамки объекта анимационного пути.

y - **(Ц)**, **Y**-координата заданной точки на векторной линии анимационного пути, относительно центра ограничительной рамки объекта анимационного пути.

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

Позволяет получить длину векторной линии указанного объекта анимационного пути.

Синтаксис:

GetTotalLength()

Возврат:

(Ч), длина (в пикселях) векторной линии анимационного пути.

Пример:

```
Debug.trace("Длина пути в пикселях: "+Path1.GetTotalLength())
```

FollowPath

Позволяет анимировать указанный графический объект вдоль анимационного пути.

Синтаксис:

FollowPath(Path,Time,Relative,Orientation,Start,End,Wait)

Параметры:

Path - **(О)**, анимационный путь следования объекта. **ОП**.

Time - **(Ч),** количество секунд, которое требуется при следовании по пути. **НП**, по умолчанию **0.0** секунд.

Relative - **(Б)**, движение указанного объекта (**true** - относительно текущей позиции объекта, **false** - со следованием по траектории абсолютно). **НП**, по умолчанию **true**.

Orientation - **(Б)**, ориентация указанного объекта вдоль пути (**true** - выровненная по траектории, **false** - фиксированная). **НП**, по умолчанию **false**.

Start - **(Ч)**, процент от **0** до **100**, начальная позиция для указанного объекта вдоль пути. **НП**, по умолчанию **0**.

End - **(Ч)**, процент от **0** до **100**, конечная позиция для указанного 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

Позволяют установить прозрачность для указанного графического объекта.

Синтаксис:

SetTransparency(Trans,Time,Wait)
Fade(Trans,Time,Wait)

Параметры:

Trans - **(Ц)**, процент прозрачности для указанного объекта (**0** - полная непрозрачность, **100** - полная прозрачность). **SetTransparency** поменяет текущее значение процента прозрачности объекта, на такое, какое указано в параметре, а **Fade** увеличит/уменьшит текущий процент прозрачности 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 полностью непрозрачно
Vector1.SetTransparency(90,10) //изображение станет прозрачным на 90% через 10 секунд
Vector1.Fade(-100,10) //прозрачность изображения через 10 секунд снизится на 80% до значения 10%
```

SetRotation и Rotate

Позволяют повернуть указанный 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 указанного объекта в градусах. **SetRotation** изменит текущий угол поворота объекта, на тот, что указан в параметре, а **Rotate** добавит к текущему углу поворота 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 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 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

Позволяют повернуть указанный 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 указанного объекта в градусах. **SetRoll** изменит текущий угол поворота объекта, на тот, что указан в параметре, а **Roll** добавит к текущему углу поворота объекта столько градусов, сколько указано в параметре. Отрицательные значения используются только для **Roll**, чтобы направление вращения было вниз. **ОП**.
Direction - **(Б)**, направление вращения указанного объекта (**true** - вверх, **false** - вниз). **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 повернуто
Vector1.SetRoll(180,false,10) //изображение перевернется сверху вниз, направление вращения вниз
Vector1.Roll(180,10) //изображение перевернется сверху вниз, направление вращения вверх
```

SetSpin и Spin

Позволяют повернуть указанный 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 указанного объекта в градусах. **SetSpin** изменит текущий угол поворота объекта, на тот, что указан в параметре, а **Spin** добавит к текущему углу поворота объекта столько градусов, сколько указано в параметре. Отрицательные значения используются только для **Spin**, чтобы направление вращения было влево. **ОП**.
Direction - **(Б)**, направление вращения указанного объекта (**true** - вправо, **false** - влево). **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 повернуто
Vector1.SetSpin(180,false,10) //изображение перевернется слева направо, направление вращения влево
Vector1.Spin(180,10) //изображение перевернется слева направо, направление вращения вправо
```

SetScale, SetScaleH, SetScaleV и Scale, ScaleH, ScaleV

Позволяют масштабировать указанный графический объект. Масштабирование происходит из центра объекта.
Изначальный масштаб 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 указанного объекта (например **1.5** - это масштаб **150%**, а **0.25** - это масштаб **25%**). **SetScale** и **SetScaleH** изменят текущий горизонтальный коэффициент масштабирования объекта, на тот, что указан в параметре, а **Scale** и **ScaleH** добавят к текущему горизонтальному коэффициенту масштабирования 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 указанного 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

Позволяют перекашивать указанный 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 указанного объекта (от **-200** до **200**, где **0** - отсутствие перекоса). **SetSkew** и **SetSkewH** поменяют текущее значение процента горизонтального перекоса объекта, на такое, какое указано в параметре, а **Skew** и **SkewH** увеличат/уменьшат текущий процент горизонтального перекоса объекта на столько процентов, сколько указано в параметре. **ОП**.
Vertical - **(Ч)**, процент вертикального перекоса для указанного 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) //перекос объекта по горизонтали и вертикали станет -200% за 10 секунд
Vector1.Skew(200,200,10) //изображение через 10 секунд перестанет быть перекошенным
```

StopAnimation и ResetAnimation

Позволяют остановить/сбросить выбранные анимации действий для указанного графического объекта.

Синтаксис:

StopAnimation(StopCode,Wait)
ResetAnimation(StopCode,Time,Wait)

Параметры:

StopCode - **(С)**, один или несколько, разделенных символом **"|",** кодов анимации для остановки/сброса. **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

Позволяет получить информацию о состоянии прокрутки указанного графического объекта. Для прокрутки графического объекта следует в свойствах 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 графического объекта.
vpos - **(Ч)**, процентная позиция вертикальной прокрутки.
hscrollbar - **(О)**, горизонтальная полоса прокрутки с ползунком, если существует, иначе **undefined**.
hvisible - **(Ч)**, процент от настоящей ширины графического 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

Позволяет изменить позицию прокрутки указанного 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

Позволяет получить объект внешнего вида для указанного графического объекта в определенном состоянии. К объекту внешнего вида можно применить следующие функции: **SetImage**, **RemoveImage**, **SetBackground**, **RemoveBackground**, **SetTexture**, **RemoveTexture**, **SetAlpha**, **RemoveAlpha**, **SetShadow**, **RemoveShadow**, **SetFlare**, **RemoveFlare**, **SetBorder**, **RemoveBorder**, **SetBtnColour**, **RemoveBtnColour**. Объект внешнего вида одного графического объекта нельзя применить к другому графическому объекту.

Синтаксис:

GetAppearance(Appearance)

Возврат:

(О) со свойствами внешнего вида конкретного графического объекта в указанном состоянии.

Параметры:

Appearance - **(С)**, название состояния указанного графического объекта. **НП**, по умолчанию **"Default"**.

Возможные состояния графического объекта:

"Default" - обычное.

"Mouse Over" - над объектом находится курсор мыши.

"Mouse Pressed" - на объекте нажата левая кнопка мыши.

"Disabled" - объект выключен.

Пример:

```
V1State1=Frame1.GetAppearance("Default")
V1State2=Frame1.GetAppearance("Mouse Over")
V1State3=Frame1.GetAppearance("Mouse Pressed")
V1State4=Frame1.GetAppearance("Disabled")
```

SetImage

Позволяет установить фоновое растровое изображение для указанного графического объекта (цвет, выбранный как прозрачный, сбрасывается).

Синтаксис:

SetImage(Filename)

Параметры:

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

Пример:

```
Frame1.SetImage(SYSTEM_PUBLICATION_DIR+"\\Picture.png")
```

RemoveImage

Позволяет убрать текущее фоновое растровое изображение для указанного графического объекта.

Синтаксис:

RemoveImage()

Пример:

```
Frame1.RemoveImage()
```

ReloadImage

Позволяет перезагрузить текущее фоновое растровое изображение для указанного графического объекта (бесполезная функция).

Синтаксис:

ReloadImage()

Пример:

```
Frame1.ReloadImage()
```

SetBackground

Позволяет создать цветной фон для указанного графического объекта. Цветной фон находится позади фонового растрового изображения.

Синтаксис:

var PropertyObject=new Object() //создание объекта, в котором будут храниться все свойства фона
PropertyObject.propertyName=value //присвоение указанному свойству фона значения
GraphicObject.SetBackground(PropertyObject) //создание фона для указанного графического объекта

Пояснение:

GraphicObject - **(О)** графический, для которого создается фон.
propertyName - название свойства создаваемого фона.
value - значение, присваиваемое указанному свойству создаваемого фона.

Параметры:

PropertyObject - **(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)**, 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

Позволяет убрать текущий цветной фон для указанного графического объекта.

Синтаксис:

RemoveBackground()

Примеры:

```
Frame1.RemoveBackground()
```

SetTexture

Позволяет создать эффект текстуры для указанного 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

Позволяет убрать текущий эффект текстуры для указанного графического объекта.

Синтаксис:

RemoveTexture()

Пример:

```
Vector1.RemoveTexture()
```

SetAlpha

Позволяет создать альфа-эффект (градиент прозрачности) для указанного графического объекта.

Синтаксис:

SetAlpha(Style,Transparency1,Transparency2)

Параметры:

Style - **(С)**, название стиля альфа-эффекта, возможные значения: **"None"**, **"Horizontal"**, **"Vertical"**, **"NWSE"**, **"NESW"**, **"Centre"**, **"Circle"**. **ОП**.

Transparency1 - **(Ц)** от **0** до **100,** процент минимальной прозрачности альфа-канала. **НП**, по умолчанию **0**.

Transparency2 - **(Ц)** от **0** до **100**, процент максимальной прозрачности альфа-канала. **НП**, по умолчанию **100**.

Пример:

```
Vector1.SetAlpha("Horizontal",0,100)
```

RemoveAlpha

Позволяет убрать текущий альфа-эффект для указанного графического объекта.

Синтаксис:

RemoveAlpha()

Пример:

```
Vector1.RemoveAlpha()
```

SetShadow

Позволяет создать эффект тени для указанного 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:

```
CLR=RGB(253,250,0);Vector1.SetShadow(5,3,50,CLR,3)
```

RemoveShadow

Позволяет убрать текущий эффект тени для указанного графического объекта.

Синтаксис:

RemoveShadow()

Пример:

```
Vector1.RemoveShadow()
```

SetFlare

Позволяет создать вокруг указанного 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:

```
CLR=RGB(50,55,155);Vector1.SetFlare(5,5,50,CLR,5)
```

RemoveFlare

Позволяет убрать текущия эффект свечения для указанного графического объекта.

Синтаксис:

RemoveFlare()

Пример:

```
Vector1.RemoveFlare()
```

SetBorder

Позволяет создать эффект рамки для указанного 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:

```
CLR=RGB(0,0,255);Vector1.SetBorder("Plain",true,5,15,CLR)
```

RemoveBorder

Позволяет убрать текущий эффект рамки для указанного графического объекта.

Синтаксис:

RemoveBorder()

Пример:

```
Vector1.RemoveBorder()
```

SetBtnColour

Позволяет создать эффект кнопки для указанного графического объекта.

Синтаксис:

SetBtnColour(SurfaceColour,LightBevelColour,DarkBevelColour,BevelWidth,BevelOpacity,BlackOutline)

Параметры:

SurfaceColour - **(Ц)** или **(С)**, цвет поверхности кнопки. **НП**, по умолчанию серый.

LightBevelColour - **(Ц)** или **(С)**, цвет светлого скоса кнопки. **НП**, по умолчанию белый.

DarkBevelColour - **(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(84,61,135);CLR2=RGB(103,83,150);CLR3=RGB(52,13,113)
Vector1.SetBtnColour(CLR1,CLR2,CLR3,10,100,false)
```

RemoveBtnColour

Позволяет убрать текущий эффект кнопки для указанного графического объекта.

Синтаксис:

RemoveBtnColour()

Пример:

```
Vector1.RemoveBtnColour()
```

SetFocus

Позволяет сфокусировать весь ввод с клавиатуры на указанном графическом объекте.

Синтаксис:

SetFocus()

Пример:

TextInput1.SetFocus()

ClearInputFocus

Позволяет отменить фокусировку ввода с клавиатуры на указанном графическом объекте.

Синтаксис:

ClearInputFocus()

Пример:

TextInput1.ClearInputFocus()

CaptureMouse

Позволяет сфокусировать весь ввод мышью на указанном графическом объекте.

Синтаксис:

CaptureMouse()

Пример:

Button1.CaptureMouse()

ReleaseMouse

Позволяет отменить фокусировку ввода мышью на указанном графическом объекте.

Синтаксис:

ReleaseMouse()

Пример:

Button1.ReleaseMouse()

CopyToClipboard

Позволяет 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**.

Image1.CopyToClipboard()

PrintObject

Позволяет распечатать указанный графический объект.

Синтаксис:

PrintObject(PrintDialog,CancelDialog)

Параметры:

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

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

Пример:

Image1.PrintObject() //распечатка изображения

Текстовые объекты

Функции текстовых объектов позволяют использовать скрипт для управления текстовыми объектами 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 шевроны <>.
3) Если данная переменная будет переобъявлена в скрипте с помощью **var**, то её отслеживание в текстовом объекте прекратится.

Функции:

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 в текстовом объекте.
LineIndex получить номер индексной позиции первого символа в нужной строчке текстового объекта.
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 диапазон символов в текстовом объекте.
SelectParagraph выбрать текст указанного абзаца в текстовом объекте.
ReplaceSelection заменить текущий выбор указанной строкой.
GetSelectionText получить выделенный текст в виде строки.
GetSelection получить положение выбранных символов.
FindText найти текстовый фрагмент в указанном текстовом объекте.
FindTextInSelection найти текстовый фрагмент только в выделенной 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 строку.
Scroll выполнить прокрутку текста.
CreateHypertext создать гипертекстовую ссылку и действие для выбора.
GetFirstHypertext получить первую гипертекстовую ссылку в текстовом объекте.
GetNextHypertext получить следующую гипертекстовую ссылку в текстовом объекте.
GetText получить строковое значение текста гиперссылки.
GetNumberHypertext получить общее количество ссылок в текстовом объекте.
RemoveAllHypertext удалить все гипертекстовые ссылки в текстовом объекте.
RestoreOriginalText вернуть содержимое текстового объекта в первоначальное состояние.
PlayAutonarrate начать автоповествование текста.
StopAutonarrate завершить автоповествование екста.
IsAutonarratePlaying проверить, работает ли в данный момент автоповествование.

GetParagraphCount

Позволяет получить количество абзацев в указанном текстовом объекте. Фрагмент текста считается за абзац после каждого перехода на новую строчку (с помощью клавиши **Enter**).

Синтаксис:

GetParagraphCount()

Возврат:

(Ц), количество абзацев в указанном текстовом объекте.

GetLineCount

Позволяет получить количество строчек в указанном текстовом объекте.

Синтаксис:

GetLineCount()

Возврат:

(Ц), количество строчек в указанном текстовом объекте.

GetWordCount

Позволяет получить количество слов в указанном текстовом объекте.

Синтаксис:

GetWordCount()

Возврат:

(Ц), количество слов в указанном текстовом объекте.

GetTextLength

Позволяет получить количество символов в указанном текстовом объекте.

Синтаксис:

GetTextLength()

Возврат:

(Ц), количество символов в указанном текстовом объекте, включая пробелы и переходы на следующую строчку.

ParagraphLength

Позволяет получить количество символов в определенном абзаце указанного текстового объекта.

Синтаксис:

ParagraphLength(ParaIndex)

Возврат:

(Ц), количество символов в указанном 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

Позволяет получить количество символов в определенной строчке указанного текстового объекта.

Синтаксис:

LineLength(LineIndex)

Возврат:

(Ц), количество символов в указанной 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

Позволяет получить номер индексной позиции первого символа указанного абзаца в текстовом объекте.

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

Позволяет получить номер индексной позиции первого символа в указанной 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

Позволяет 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

Позволяет 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 текстового объекта.

y - **(Ч)**, **Y**-координата верхнего левого угла символа относительно текстового объекта.

width - **(Ч)**, ширина символа.

height - **(Ч)**, высота символа.

Параметры:

CharIndex - **(Ц),** номер индексной позиции символа (нумерация начинается с **0**). **ОП**.

CharFromPoint

Позволяет 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 текстового объекта. **ОП**.

PosY - **(Ц)**, **Y**-координата верхнего левого угла символа относительно текстового 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) //переменная содержит данные о первом символе 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 координатам (5,8): "+Text1.CharFromPoint(5,8))
```

SetSelection и SelectParagraph

Позволяют выбрать фрагмент текста в указанном 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 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

Позволяет 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

Позволяет получить, выбранный с помощью функции **SetSelection** или **SelectParagraph**, фрагмент текста как строковое значение.

Синтаксис:

GetSelectionText()

Возврат:

(С), выбранный с помощью функции **SetSelection** или **SelectParagraph**, фрагмент текста. Если выбранного фрагмента нет, то строковое значение будет пустым.

GetSelection

Позволяет 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 текстовом объекте
Text1.SetSelection(1,1) //выбор между первым и вторым символом в текстовом объекте
Text1.SetSelection(0,1) //выбор первого символа в текстовом объекте
Text1.SetSelection(1,2) //выбор второго символа в текстовом объекте
Text1.ReplaceSelection("#")
Debug.trace("Выбранный текст: "+Text1.GetSelectionText())
var txtSlctn1=Text1.GetSelection()
Debug.trace("\n"+"Номера позиций выбранного фрагмента: "+txtSlctn1.start+","+txtSlctn1.end)
```

FindText и FindTextInSelection

Позволяют найти символ, слово или фразу в текстовом объекте с указанной позиции. **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). Поиск зависит от регистра символов.

Синтаксис:

FindText(String,StartPosition)
FindTextInSelection(String,StartPosition)

Возврат:

(Ц), номер индексной позиции первого символа искомого текста, если он найден, и **-1**, если не найден.

Параметры:

String - **(С)** для поиска в текстовом объекте. **ОП**.
StartPosition - **(Ц)**, номер индексной позиции символа, с которой начнётся поиск (нумерация начинается с **0**). **НП**, по умолчанию **0**.

Пример:

```
var Word1="Тапир" //объявлена переменная, содержащая слово для поиска
var WordSearch1=0 //объявлена переменная для хранения количества найденных слов
var FindResult=Text1.FindText(Word1,0) //первый поиск слова с начала текста
if (FindResult!=-1) {WordSearch1=WordSearch1+1} //если слово найдено то к счетчику прибавляется 1
while (FindResult!=-1) //цикл для поиска оставшихся слов если первое слово было найдено
{//начало цикла
var FindResult=Text1.FindText(Word1,FindResult+1) //поиск слова с позиции после найденного
if (FindResult!=-1) {WordSearch1=WordSearch1+1} //если слово найдено то к счетчику прибавляется 1
} //конец цикла
Debug.trace("Найдено: "+WordSearch1) //вывод количества найденных слов в скриптовую консоль
```

SetColour

Позволяет изменить цвет указанного 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);NEWcolor=RGB(253,250,0);Text1.SetColour(NEWcolor)
wait(1);Text1.SetColour(RGB(0,170,65))
wait(1);Text1.SetColour(-1)
```

GetSelectionStyle

Позволяет 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 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

Позволяет назначить 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) //выбран весь текст первого текстового объекта
var Text1Style=Text1.GetSelectionStyle() //в переменной будут храниться все свойства стиля 1-го текста
//вывод значений всех свойств стиля 1-го текста
Debug.trace("Жирность: "+Text1Style.bold+"\n")
Debug.trace("Курсив: "+Text1Style.italic+"\n")
Debug.trace("Подчеркивание: "+Text1Style.underline+"\n")
Debug.trace("Нижний индекс: "+Text1Style.subscript+"\n")
Debug.trace("Верхний индекс: "+Text1Style.superscript+"\n")
Debug.trace("Название шрифта: "+Text1Style.fontname+"\n")
Debug.trace("Размер шрифта: "+Text1Style.fontsize+"\n")
Debug.trace("Аспект шрифта: "+Text1Style.fontaspect+"\n")
Debug.trace("Цвет текста: "+Text1Style.colour+"\n")
Debug.trace("Цвет фона текста: "+Text1Style.backgroundcolour+"\n")
Debug.trace("Цвет тени текста: "+Text1Style.shadowcolour+"\n")
wait(3) //пауза на три секунды
var NewText1Style=new Object() //переменной присвоено, что она новый объект
NewText1Style.colour="FFFF00" //новому объекту присвоено свойство цвета со значением желтого
Text1.SetSelectionStyle(NewText1Style) //первому тексту назначено новое свойство: желтый цвет
Text2.SetSelection(0,-1) //выбран весь текст второго текстового объекта
Text2.SetSelectionStyle(Text1Style) //2-му тексту присвоены все старые свойства стиля 1-го текста
```

GetSelectionParagraphStyle

Позволяет 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 paraNum=0
var paraBegin=Text1.ParagraphIndex(paraNum)
var paraEnd=paraBegin+Text1.ParagraphLength(paraNum)-1
Text1.SetSelection(paraBegin,paraEnd)
//Получение всех свойств выбранного абзаца и вывод их значений в скриптовой консоли
var ParaStyle=Text1.GetSelectionParagraphStyle()
Debug.trace("Выравнивание: "+ParaStyle.justification+"\n")
Debug.trace("Междустрочный интервал: "+ParaStyle.linespacing+"\n")
Debug.trace("Разрыв между текущим абзацем и предыдущим: "+ParaStyle.spacingbefore+"\n")
Debug.trace("Разрыв между текущим абзацем и следующим: "+ParaStyle.spacingafter+"\n")
Debug.trace("От левого края до абзаца: "+ParaStyle.leftindent+"\n")
Debug.trace("От правого края до абзаца: "+ParaStyle.rightindent+"\n")
Debug.trace("Отступ первой строчки абзаца: "+ParaStyle.firstlineindent+"\n")
Debug.trace("Выступ абзаца: "+ParaStyle.hangingindent+"\n")
Debug.trace("Тип маркера: "+ParaStyle.bullet.type+"\n")
Debug.trace("Символ маркера: "+ParaStyle.bullet.character+"\n")
Debug.trace("Шрифт маркера: "+ParaStyle.bullet.font+"\n")
Debug.trace("Цвет маркера: "+ParaStyle.bullet.colour+"\n")
```

SetSelectionParagraphStyle

Позволяет назначить в текстовом объекте стиль для абзацев, чьи части входят в, выбранный с помощью функции **SetSelection** или **SelectParagraph**, фрагмент текста. 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 paraNum=0;var paraBegin=Text1.ParagraphIndex(paraNum)
var paraEnd=paraBegin+Text1.ParagraphLength(paraNum)-1
Text1.SetSelection(paraBegin,paraEnd)
//Изменение свойств стиля абзаца
var ParaStyle=Text1.GetSelectionParagraphStyle() //в переменной содержатся все свойства стиля абзаца
var NewParaStyle =new Object() //создание нового объекта для нового стиля абзаца
NewParaStyle.justification="Centre" //свойство выравнивания в новом стиле: по центру
Text1.SetSelectionParagraphStyle(NewParaStyle) //выдел. абзацу в 1 тексте назначено новое свойство
wait(3)  //пауза на три секунды
Text2.SetSelection(0,-1) //выделен весь текст во 2 текстовом объекте
Text2.SetSelectionParagraphStyle(ParaStyle) //присв. 2 тексту всех свойств старого стиля абзаца из 1
```

SetListBoxSelection

Позволяет 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**.

Примеры:

```
Listbox1.SetListBoxSelection(1) //выделяется вторая строчка списка
```

Scroll

Позволяет прокручивать текст в указанном 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, не требуется для команд **"ParagraphUp"**, **"ParagraphDown"**, **"PageUp"**, **"PageDown"** (их прокрутка всегда только на один абзац/страницу) и **"Start"**, **"End"** (прокрутка в начало/конец).

Для команд с **"LineUp"** и **"LineDown"** - число строчек для прокрутки. **НП**, по умолчанию **1**.

Для команды с **"LineTo"** - номер строчки (нумерация начинается с **0**), куда следует совершить прокрутку. **НП**, по умолчанию **0**.

Пример:

```
Text1.Scroll("ParagraphDown") //прокрутка текста на один абзац вниз
Text1.Scroll("LineTo",3) //прокрутка текста к четвертой строчке
Text1.Scroll("End") //прокрутка в конец текста
```

CreateHypertext

Позволяет 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)**, фрагмент кода, который выполнится при нажатии левой кнопкой мыши на гиперссылку. **ОП**.

ВАЖНО: Если пользовательская функция без параметров, то можно указать в параметре **Action** только её название без скобок, и не как строковое значение. Если пользовательская функция содержит параметры, то её следует указать в параметре **Action** как фрагмент кода в виде строкового значения.

GetFirstHypertext

Позволяет получить первую (самую ближнюю к концу текста) гиперссылку в указанном текстовом объекте.

Синтаксис:

GetFirstHypertext()

Возврат:

(О), первая гиперссылка или **null**, если в текстовом объекте гиперссылок нет.

GetNextHypertext

Позволяет получить следующую гиперссылку в указанном 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

Позволяет получить текст, содержащийся в гиперссылке, как строковое значение.

Синтаксис:

GetText()

Возврат:

(С), текст, содержащийся в гиперссылке.

GetNumberHypertext

Позволяет получить количество гиперссылок в указанном текстовом объекте.

Синтаксис:

GetNumberHypertext()

Возврат:

(Ц), количество гиперссылок в указанном текстовом объекте.

Общий пример:

```
//пользовательские функции задаются только в скриптовом объекте страницы
function Zveronica() {Text1.SetColour("yellow")} //создание пользовательской функции без параметров
Text1.SetSelection(2,8) //выделение фрагмента текста для гиперссылки
Text1.CreateHypertext(Zveronica) //функция выполнится при нажатии на гиперссылку
Text1.SetSelection(18,24) //выделение фрагмента текста для еще одной новой гиперссылки
Text1.CreateHypertext('Text1.SetColour("red")') //код выполнится при нажатии на новую гиперссылку
var Link1=Text1.GetFirstHypertext() //переменная содержит объект первой гиперссылки
Debug.trace("Текст первой гиперссылки: "+Link1.GetText()+"\n")
var Link2=Text1.GetNextHypertext(Link1) //переменная содержит объект второй гиперссылки
Debug.trace("Текст второй гиперссылки: "+Link2.GetText()+"\n")
Debug.trace("Количество гиперссылок: "+Text1.GetNumberHypertext())
```

RemoveAllHypertext

Позволяет убрать все гиперссылки в указанном текстовом объекте.

Синтаксис:

RemoveAllHypertext()

Пример:

```
Text1.RemoveAllHypertext()
```

RestoreOriginalText

Позволяет вернуть содержимое указанного текстового объекта к исходному состоянию.

Синтаксис:

RestoreOriginalText()

Пример:

```
Text1.RestoreOriginalText()
```

PlayAutonarrate

Позволяет начать воспроизведение автоповествования указанного текстового объекта.

Синтаксис:

PlayAutonarrate()

Пример:

```
Text1.PlayAutonarrate()
```

StopAutonarrate

Позволяет остановить воспроизведение автоповествования указанного текстового объекта.

Синтаксис:

StopAutonarrate()

Пример:

```
Text1.StopAutonarrate
```

IsAutonarratePlaying

Позволяет проверить, воспроизводится ли сейчас автоповествование указанного текстового объекта.

Синтаксис:

IsAutonarratePlaying()

Возврат:

(Б), состояние воспроизведения автоповествования (**true** - воспроизводится, **false** - нет).

Пример:

```
Debug.trace("Проверка работы автоповествования: "+Text1.IsAutonarratePlaying())
```

Кнопки

Функции используются для определения и изменения состояния кнопки.
Настройка кнопки 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 включатель/выключатель.
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.
Функции:
GetState получить текущее состояние нажатости кнопки.
SetState установить состояние нажатости кнопки.
GetTextObject получить текстовый объект, находящийся внутри кнопки.

GetState

Позволяет получить текущее состояние "нажатости" указанного объекта 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

Позволяет установить состояние "нажатости" указанного объекта 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

Позволяет 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 кнопки.
Пример:

```
BtnText=Button1.GetTextObject()
while (true) {wait(3);BtnText.SetColour(RGB(0,170,65))}
```

Слайд-шоу

Функции объекта слайд-шоу позволяют управлять, содержащимися внутри слайдами (растровыми изображениями). Поддерживаемые форматы: **BMP**, **CGM**, **JPG**, **JPEG**, **PCX**, **PNG**, **TGA**, **GIF**, **TIF**, **TIFF**, **PCD**, **WMF**, **EMF**, **ILV**.

Функции:

Play воспроизвести слайд-шоу.
Stop остановить а 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

Позволяет начать воспроизведение указанного слайд-шоу с определенного слайда.

Синтаксис:

Play(Slide)

Параметры:

Slide - **(Ц)**, номер начального слайда (нумерация начинается с **0**) для воспроизведения. **НП**, по умолчанию **0**.

Пример:

```
Slideshow1.Play()
```

Stop

Позволяет остановить воспроизведение указанного слайд-шоу. Слайд-шоу останавливается на текущем слайде и не сбрасывается на первый слайд в слайд-шоу.

Синтаксис:

Stop()

Пример:

```
Slideshow1.Stop()
```

Pause

Позволяет приостановить воспроизведение указанного слайд-шоу.

Синтаксис:

Pause()

Пример:

```
Slideshow1.Pause()
```

Continue

Позволяет продолжить воспроизведение указанного слайд-шоу. продолжить с текущей позиции в слайд-шоу, когда слайд-шоу было остановлено или приостановлено.

Синтаксис:

Continue()

Пример:

```
Slideshow1.Continue()
```

IsPlaying

Позволяет 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()) //проверка работы слайд-шоу
```

GotoSlide

Позволяет перейти к определенному слайду в указанном слайд-шоу.

Синтаксис:

GotoSlide(Slide)

Параметры:

Slide - **(Ц)**, номер слайда для перехода (нумерация начинается с **0**), **-1** для перехода на последний слайд. **ОП** .

Пример:

```
Slideshow1.GotoSlide(3) //перейти на четвертый слайд
```

GetSlide

Позволяет получить порядковый номер текущего слайда в указанном слайд-шоу.

Синтаксис:

GetSlide()

Возврат:

(Ц), порядковый номер текущего слайда (нумерация начинается с **0**).

Пример:

```
Slideshow1.GetSlide()
```

GetSlideCount

Позволяет получить количество слайдов в указанном слайд-шоу.

Синтаксис:

GetSlideCount()

Возврат:

(Ц), количество слайдов в указанном слайдшоу.

Пример:

```
Debug.trace("Количество слайдов: "+Slideshow1.GetSlideCount())
```

UpdateFiles

Позволяет 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 позволяют рисовать внутри векторные полигоны.

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) Для рисования нескольких независимых фигур, используется 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.

Функции:

DrawLine нарисовать линию.
DrawRectangle нарисовать прямоугольник.
DrawRoundRectangle нарисовать прямоугольник со скругленными углами.
DrawEllipse нарисовать окружность или эллипс.
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 формате **WMF**, 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.

Функции:

SetLineColour назначить цвет линии полигона.
SetFillColour назначить цвет заливки полигона.

DrawLine

Позволяет рисовать линии.

Синтаксис:

DrawLine(StartX,StartY,EndX,EndY)

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**.

Если есть две переменные, каждая из которых содержит значения сразу двух координат (например: они получены с помощью функции **GetPosition()** и подобных), то вместо четырех параметров, можно поставить два со значениями этих переменных.

Пример:

```
Frame1.DrawLine(0,0,300,300) //рисуется линия по четырем параметрам
var Start=Frame1.GetPosition() //создается переменная с двумя координатами для начала линии
Start.x=0;Start.y=300 //переназначение координат начала
var End=Frame1.GetPosition() //создается переменная с двумя координатами для конца линии
End.x=300;End.y=0 //переназначение координат конца
Frame1.DrawLine(Start,End) //рисуется линия по двум параметрам
```

DrawRectangle

Позволяет рисовать прямоугольники.

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:

```
Frame1.DrawRectangle(0,0,200,400) //рисуется прямоугольник
```

DrawRoundRectangle

Позволяет рисовать прямоугольники 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:

```
Frame1.DrawRoundRectangle(0,0,400,200,40) //рисуется прямоугольник со скругленными углами
```

При рисовании фигуры эллипса, звезды или многоугольника учитываются не координаты её центра, а координаты верхнего левого угла воображаемого прямоугольника, в который фигура вписана. Такой прямоугольник называется ограничительным. Многоугольники с нечётным числом сторон и звезды с нечётным числом лучей рисуются со смещением вверх от ограничительного прямоугольника (особенно ярко выражено у треугольников, пятиугольников и трёхконечных, пятиконечных звезд). Для того, чтобы убрать смещение, к параметру **PosY** следует прибавить значение: для треугольников и трехконечных звезд **Hight*0.165**, для пятиугольников и пятиконечных звезд **Hight*0.05**.

DrawEllipse

Позволяет рисовать окружности и эллипсы.

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:

```
Frame1.DrawEllipse(50,10,100,200,0) //рисуется эллипс
Frame1.DrawEllipse(50,10,100,200,3) //рисуется синусоидальная волна
```

DrawStar

Позволяет рисовать ромбы и многоконечные звезды.

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:

```
Frame1.DrawStar(200,10,200,200,2) //рисуется ромб
Frame1.DrawStar(200,10,200,200,8) //рисуется восьмиконечная звезда
```

DrawRegPoly

Позволяет рисовать многоугольники.

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:

```
Frame1.DrawRegPoly(20,250+150*0.165,150,150,3) //рисуется треугольник
Frame1.DrawRegPoly(20,250,150,150,6) //рисуется шестиугольник
```

AddPoint

Позволяет добавлять точки в полигон, нарисованный с помощью скрипта.

Синтаксис:

AddPoint(Index,PosX,PosY)

Параметры:

Index - **(Ц)**, порядковый номер существующей точки для добавления новой точки после нее. Нумерация точек начинается с **0**. Для добавления новой точки, следующей после последней точки полигона, используется как значение номера последней точки, так и значение номера равное **-1**. **НП**, по умолчанию **-1**.

PosX - **(Ц)**, **X**-координата новой точки. **ОП**.

PosY - **(Ц)**, **Y**-координата новой точки. **ОП**.

Если есть переменная, содержащая значения сразу двух координат (например: полученная с помощью функции **GetPosition()** и подобных), то вместо двух параметров **PosX** и **PosY**, можно поставить один с её значением.

MovePoint

Позволяет перемещать существующие точки полигона, нарисованного с помощью скрипта.

Синтаксис:

MovePoint(Index, PosX, PosY)

Параметры:

Index - **(Ц)**, порядковый номер перемещаемой существующей точки. Нумерация точек начинается с **0**. Для перемещения последней точки полигона, используется как значение её номера, так и значение номера равное **-1**. **НП**, по умолчанию **-1**.

PosX - **(Ц)**, новая **X**-координата перемещаемой точки. **ОП**.

PosY - **(Ц)**, новая **Y**-координата перемещаемой точки. **ОП**.

Если есть переменная, содержащая значения сразу двух координат (например: полученная с помощью функции **GetPosition()** и подобных), то вместо двух параметров **PosX** и **PosY**, можно поставить один с её значением.

RemovePoint

Позволяет удалять точки из полигона, нарисованного с помощью скрипта.

Синтаксис:

RemovePoint(Index)

Параметры:

Index - (**Ц**), порядковый номер удаляемой существующей точки. Нумерация точек начинается с **0**. Для удаления последней точки полигона, используется как значение её номера, так и значение номера равное **-1** . **НП**, по умолчанию **-1**. Точку под номером **0** удалить нельзя.

ClearDraw

Позволяет очистить кадр от всех полигонов, нарисованных с помощью скрипта.

Синтаксис:

ClearDraw()

Общий пример:

```
var Pos=Frame1.GetPosition() //переменной присваиваются координаты кадра (его центр)
Frame1.DrawLine(200,0,200,400) //рисуется вертикальная линия сверху вниз
Frame1.RegisterEventHandler("lclick",this.MoveHandler) //регистрация события
function MoveHandler(Click) //функция перемещения точки по клику мышки
{this.MovePoint(0,Click.x,Click.y)} //перемещение первой точки туда где сделан левый клик мышью
Frame1.AddPoint(-1,400,200) //из конца предыдущей линии продолжается кривая вверх вправо
Frame1.AddPoint(-1,Pos) //кривая продолжается в центр кадра
wait(3) //первая пауза в три секунды
Frame1.RemovePoint() //удаляется в центре кадра точка кривой
wait(3) //вторая пауза в три секунды
Frame1.ClearDraw() //очистка кадра от всех нарисованных скриптом полигонов
```

SetLineStyle

Позволяет назначить ширину и тип линий для всех полигонов кадра, нарисованных с помощью скрипта.

Синтаксис:

SetLineStyle(Width,Style,Dash,Shift)

Параметры:

Width - 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**.

Примеры:

```
Frame1.SetLineStyle(4,2,16) //линии всех полигонов кадра станут штриховыми
```

SetLineColour и SetFillColour

Эти функции работают как с полигонами, созданными с помощью скрипта, так и с полигонами нарисованными в векторном редакторе **Opus Pro** или импортированными. При использовании функции с полигонами, нарисованными скриптом, цвет линии/заливки применяется на все фигуры в кадре, поэтому чтобы сделать полноценное разноцветное изображение, следует создать несколько кадров. При использовании функции с полигонами, нарисованными в редакторе или импортированными, цвет линии/заливки применяется к конкретному полигону и на другие не распространяется.

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**.

Пример:

```
//Заливка всех полигонов кадра разными цветами, по очереди, разными способами
wait(2);Frame1.SetFillColour("red",5)
wait(2);Frame1.SetFillColour("#FFA500",10)
wait(2);Frame1.SetFillColour(RGB(255,237,0),15)
wait(2);NEWcolor=RGB(0,255,0)
Frame1.SetFillColour(NEWcolor,20)
wait(2);Frame1.SetFillColour(0,119,201,25)
var red=0;var green=69;var blue=146
wait(2);Frame1.SetFillColour(red,green,blue,30)
wait(2);Frame1.SetFillColour(-1) //удаление заливки всех полигонов кадра
```

Мультикадры

Мультикадры - графические объекты, содержащие внутри себя несколько объектов кадров, между которыми можно переключаться. События, происходящие в отдельных кадрах не сбрасываются при переключении кадров.

Функции:
Play начать проигрывание кадров друг за другом.
Stop остановить проигрывание кадров.
Forward перейти к следующему кадру.
Backward перейти к предыдущему кадру.
ToStart перейти к начальному кадру.
ToEnd перейти к последнему кадру.
ToFrame перейти к указанному кадру.
ToRandom перейти к случайному кадру.

Play

Позволяет 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() //запустить переключение кадров в мультикадре

Stop

Позволяет остановить на текущем кадре проигрывание указанного мультикадра.

Синтаксис:
Stop()
Пример:
MultiFrame1.Stop() //остановить переключение кадров в мультикадре

Forward

Позволяет перейти указанному объекту мультикадра на следующий кадр. Если текущий кадр последний, то произойдет переход к начальному кадру.

Синтаксис:
Forward()
Пример:
MultiFrame1.Forward() //переход к следующему кадру в мультикадре

Backward

Позволяет перейти указанному объекту мультикадра на предыдущий кадр. Если текущий кадр начальный, то произойдет переход к последнему кадру.

Синтаксис:
Backward()
Пример:
MultiFrame1.Backward() //переход к предыдущему кадру в мультикадре

ToStart

Позволяет перейти указанному объекту мультикадра на начальный кадр.

Синтаксис:

ToStart()

Пример:

MultiFrame1.ToStart() //переход к первому кадру в мультикадре

ToEnd

Позволяет перейти указанному объекту мультикадра на последний кадр.

Синтаксис:

ToEnd()

Пример:

MultiFrame1.ToEnd() //переход к последнему кадру в мультикадре

ToFrame

Позволяет перейти указанному объекту мультикадра на определенный кадр.

Синтаксис:

ToFrame(Number)

Параметры:

Number - **(Ц)**, номер кадра в указанном мультикадре (нумерация начинается с **0**). **ОП**.

Пример:

MultiFrame1.ToFrame(1) //переход ко второму кадру в мультикадре

ToRandom

Позволяет перейти указанному объекту мультикадра на случайный кадр (без повторений, пока не будут показаны все кадры).

Синтаксис:

ToRandom()

Пример:

MultiFrame1.ToRandom() //переход к случайному кадру в мультикадре

Tween-анимация

Функции объектов **Tween**-анимации позволяют управлять их воспроизведением на странице публикации.
ВАЖНО: к объекту **Tween**-анимации нельзя прикреплять скриптовый объект, но к его дочерним элементам можно.

Функции:

Play воспроизвести анимацию с текущего кадра.
Stop остановить воспроизведение анимации на текущем кадре.
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.
GotoFrame перейти к указанному кадру временной шкалы.

Play

Позволяет воспроизвести объект **Tween**-анимации с текущего кадра временной шкалы.

Синтаксис:

Play()

Пример:

Tween1.Play()

Stop

Позволяет остановить воспроизведение объекта **Tween**-анимации на текущем кадре временной шкалы.

Синтаксис:

Stop()

Пример:

Tween1.Stop()

GotoAndPlay

Позволяет перейти к указанному кадру временной шкалы объекта **Tween**-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

Позволяет перейти к указанному кадру временной шкалы объекта **Tween**-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)

GotoFrame

Позволяет перейти к указанному кадру временной шкалы объекта **Tween**-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)

Таймлайны (временные шкалы)

Функции временной шкалы используются для запуска или остановки объекта временной шкалы на странице публикации.

Функции:

Start запустить воспроизведение временной шкалы.
Stop остановить воспроизведение временной шкалы.

Start

Позволяет начать воспроизведение указанного объекта временной шкалы.

Синтаксис:

Start()

Пример:

TimeLine1.Start()

Stop

Позволяет остановить воспроизведение указанного объекта временной шкалы.

Синтаксис:

Stop()

Пример:

TimeLine1.Stop()

Видео

Видеообъекты содержат видеофайлы со следующими поддерживаемыми видеоформатами: **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.

Функции:

Play воспроизвести видео.
Stop остановить воспроизведение видео.
IsPlaying проверить воспроизводится ли видео.
Go назначить стартовую позицию воспроизведения видео.
Seek перемотать видео в нужную 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

Позволяет воспроизвести указанное 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 параметры **From** и **To** не указаны, воспроизводится всё видео целиком.
Synch - **(Б)**, отправка сообщения о синхронизации (**true** - отправляется, **false** - не отправляется). **НП**, по умолчанию **true**.

Stop

Позволяет остановить воспроизведение указанного видео.

Синтаксис:

Stop(Reset,Synch)

Параметры:

Reset - **(Б)**, сброс видео к началу (**true** - видео сбрасывается к началу, **false** - видео останавливается на текущем моменте). **НП**, по умолчанию **false**.
Synch - **(Б)**, отправка сообщения о синхронизации (**true** - отправляется, **false** - не отправляется). **НП**, по умолчанию **true**.

IsPlaying

Позволяет проверить, воспроизводится ли указанное видео в данный момент или нет.

Синтаксис:

IsPlaying()

Возврат:

(Б), состояние воспроизведения видео (**true** - воспроизводится, **false** - нет).

Общий пример:

```
Video1.Play() //запуск воспроизведения видео с самого начала
Debug.trace("Играет ли видео?:"+Video1.IsPlaying()) //проверка воспроизведения видео
wait(3);Video1.Stop() //остановить видео через три секунды после запуска
Debug.trace("\n"+"Играет ли видео?:"+Video1.IsPlaying()) //проверка воспроизведения видео
wait(3);Video1.Play() //продолжение воспроизведения видео через три секунды с места остановки
Debug.trace("\n"+"Играет ли видео?:"+Video1.IsPlaying()) //проверка воспроизведения видео
```

Go

Позволяет 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 в функции **Play**, то начальная позиция воспроизведения, стоящая в функции **Go** заменится.

Синтаксис:

Go(Position)

Параметры:

Position - **(Ч)**, позиция в секундах, с которой начнется воспроизведение видео. Значение **-1** устанавливает позицию в конец видео. **ОП**.

Пример:

```
Video1.Go(3) //установка стартовой позиции воспроизведения для указанного видео
Video1.Play() //видео начнет проигрываться не с начала, а с третьей секунды
```

Seek

Позволяет перемотать указанное видео в нужную позицию.

Синтаксис:

Seek(Action,Amount)

Параметры:

Action - **(С)**, способ перемотки видео, 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. Данный параметр используется только если параметр **Action** имеет значение **"forward"** или **"backward"**.

Пример:

```
Video1.Seek("forward",3.987) //перемотка видео вперед на указанное количество секунд
Video1.Seek("end") //перемотка видео в самый конец
```

GetLength

Позволяет получить длину указанного видео.

Синтаксис:

GetLength()

Возврат:

(Ч), длина указанного видео в секундах (**-1** если не определилась).

Пример:

```
Debug.trace(Video1.GetLength()) //в скриптовой консоли выведется длина видео
```

GetLengthBytes и GetBufferLength

Позволяют получить длину указанного видео в байтах и длину видеобуфера.

Синтаксис:

GetLengthBytes()
GetBufferLength()

Возврат:

(Ч), длина указанного видео в байтах и длина видеобуфера (**-1** если не определилась).

Пример:

```
Debug.trace(Video1.GetLengthBytes()) //в скриптовой консоли выведется длина видео в байтах
Debug.trace("\n"+Video1.GetLengthBytes()) //в скриптовой консоли выведется длина видеобуфера
```

GetPosition

Позволяет получить текущую временную позицию воспроизведения видео. Для видеообъектов замещает стандартное действие функции **GetPosition**, с помощью которой можно получить координаты расположения объекта, поэтому, чтобы узнать координаты видеообъекта, используются функции **GetXPosition** и **GetYPosition**.

Синтаксис:

GetPosition()

Возврат

(Ч), текущая временная позиция воспроизведения указанного видео в секундах.

Пример:

```
Debug.trace(Video1.GetPosition()) //в скриптовой консоли выведется позиция воспроизведения видео
```

Звук и музыка

Функции звука и музыки используются для воспроизводения в публикации музыкальных **CD** и звуковых файлов со следующими поддерживаемыми форматами: **ASF**, **WMA**, **WMV**, **WAV**, **MID**, **RMI**, **MP3**, **OGG**.

Функции:

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 трека **CD** в секундах.
Stop остановить воспроизведение звукового объекта.
Seek перемотать звуковой объект в нужную 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 устройству или звуковому объекту.
GetPosition получить текущую позицию воспроизведения звукового объекта в секундах.
SetPosition назначить текущую позицию воспроизведения звукового объекта в секундах.

PlaySystemSound

Позволяет воспроизвести нужный тип системного звука.

Синтаксис:

PlaySystemSound(Type)

Параметры:

Type - **(С)**, тип системного звука для воспроизведения, возможные значения: **"Default"**, **"Asterisk"**, **"Question"**, **"Exclamation"**, **"Error"**. **ОП**.

Пример:

```
PlaySystemSound("Error") //воспроизведение системного звука об ошибке
```

OpenSound

Позволяет 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") //создание звукового объекта
```

CreateCDPlayer

Позволяет открыть музыкальный **CD** без воспроизведения и сделать его объектом **CD**-плеера.

Синтаксис:

CreateCDPlayer()

Возврат:

(О) **CD**-плеера.

Пример:

```
var CDMusic1=CreateCDPlayer() //создание объекта CD-плеера
```

PlaySound, PlayCDTrack и Play

Функция **Playsound** позволяет воспроизвести напрямую указанный звуковой файл, без функции **OpenSound** . Функция **PlayCDTrack** позволяет воспроизвести напрямую указанный трек музыкального **CD**, без функции **CreateCDPlayer**. Функция **Play** позволяет воспроизвести либо указанный звуковой объект, созданный функцией **OpenSound**, либо указанный 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 нужному звуковому файлу. **ОП**.
Track - **(Ц)**, номер трека **CD** для воспроизведения (нумерация начинается с **1**). **ОП**.
Preload - **(Б)**, предварительная загрузка звука (**true** - звук будет предзагружен, **false** - нет). **НП**, по умолчанию **false**.
Times - **(Ц)**, количество раз воспроизведения звука/трека. Для циклического воспроизведения используется значение **-1**. **НП**, по умолчанию **1**.
Volume - **(Ц)** от **0** до **100**, процентная громкость воспроизведения звука/трека. **НП**, по умолчанию **100**.
Start - **(Ч)**, позиция в секундах, с которой начнется воспроизведение звука/трека. **НП**, по умолчанию **0** (воспроизведение с самого начала).
Finish - **(Ч)**, позиция в секундах, на которой закончится воспроизведение звука/трека. **НП**, по умолчанию **-1** (воспроизведение 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)**, регуляция воспроизведение звука/трека при смене страницы (**true**- звук/трек остановится, **false** - нет). **НП**, по умолчанию **true**.
Channel - **(Ц)** от **1** до **8**, канал для воспроизведения звука. **(С)** **"any"** для воспроизведения звука любым каналом. **НП**, по умолчанию **"any"**.
Wait - **(Б)**, установка выполнения следующей строчки кода (**true** - следующая строчка кода выполнится только после завершения воспроизведения звука/трека, **false** - следующая строчка кода выполнится сразу после старта воспроизведения звука/трека). **НП**, по умолчанию **true** для звуковых объектов и **false** для треков **CD**.

Пример:

```
PlaySound(SYSTEM_PUBLICATION_DIR+"MUSIC.MP3") //воспроизведение звукового файла
PlayCDTrack(3) //воспроизведение 3 трека напрямую с музыкального компакт диска
var Sound1=OpenSound(SYSTEM_PUBLICATION_DIR+"MUSIC.MP3") //создание звукового объекта
Sound1.Play() //воспроизведение звукового объекта
var CDMusic1=CreateCDPlayer() //создание объекта CD-плеера
CDMusic1.Play(3) //воспроизведение 3 трека объекта CD-плеера
```

IsPlaying

Позволяет 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() //создание объекта CD-плеера
CDMusic1.Play(1) //воспроизведение 1 трека объекта CD-плеера
Debug.trace("Проигрывается ли CD?:"+CDMusic1.IsPlaying()) //проверка воспроизведения CD
```

GetTrackLength

Позволяет получить длину указанного трека **CD**.

Синтаксис:

GetTrackLength(Track)

Возврат:

(Ч), длина указанного трека **CD** в секундах. Если трека под указанным номером не существует, либо **CD** отсутствует в приводе, то **-1**.

Параметры:

Track - **(Ц)**, номер трека **CD** для проверки длины (нумерация начинается с **1**). **ОП**.

Пример:

```
var CDMusic1=CreateCDPlayer() //создание объекта CD-плеера
Debug.trace("Длительность:"+CDMusic1.GetTrackLength(1)+" сек.") //вывод длины 1-го трека CD
```

Stop

Позволяет остановить и сбросить воспроизведение звукового объекта.

Синтаксис:

Stop(Reset)

Параметры:

Reset - **(Б)**, последующее воспроизведение данного звукового объекта (**true** - будет с самого начала, **false** - продолжится с места остановки). **ОП**.

Пример:

```
var Sound1=OpenSound(SYSTEM_PUBLICATION_DIR+"MUSIC.MP3") //создание звукового объекта
Sound1.Play() //воспроизведение звукового объекта
wait(30);Sound1.Stop(false) //воспроизведение звука остановится через 30 секунд без сброса
wait(10);Sound1.Play() //воспроизведение звука возобновится через 10 секунд с места остановки
```

Seek

Позволяет перемотать указанный звуковой объект в нужную позицию.

Синтаксис:

Seek(Action,Amount)

Параметры:

Action - **(С)**, направление перемотки звукового объекта, возможные значения: **"forward"** (перемотать вперед), **"backward"** (перемотать назад), **"end"** (перемотать в конец), **"start"** (перемотать в начало). **ОП**.

Amount - **(Ч)**, время в секундах , указывающее на сколько следует перемотать звуковой объект вперед или назад. Данный параметр используется только если параметр **Action** имеет значение **"forward"** или **"backward"**.

Пример:

```
var Sound1=OpenSound(SYSTEM_PUBLICATION_DIR+"MUSIC.MP3") //создание звукового объекта
Sound1.Play() //воспроизведение звукового объекта
wait(10);Sound1.Stop(false) //воспроизведение звука остановится через 10 секунд без сброса
Sound1.Seek("forward",30.5) //перемотка звукового объекта на 30.5 секунд вперед с места остановки
Sound1.Play() //воспроизведение звукового объекта с места окончания перемотки
```

Pause и Resume

Позволяют приостановить и возобновить воспроизведение указанного звукового объекта. Данные функции предназначены только для форматов **WAV** и **MP3**. **Не работают с форматами OGG и MID.**

Синтаксис:

```
Pause()
Resume()
```

Пример:

```
var Sound1=OpenSound(SYSTEM_PUBLICATION_DIR+"MUSIC.WAV") //создание звукового объекта
Sound1.Play() //воспроизведение звукового объекта
wait(10);Sound1.Pause() //воспроизведение звука приостановится через 10 секунд
wait(10);Sound1.Resume() //воспроизведение звука возобновится с места паузы через 10 секунд
```

GetVolume

Позволяет получить текущий уровень громкости указанного устройства или указанного 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

Позволяет установить уровень громкости указанному устройству или указанному звуковому объекту. 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, на котором изменится громкость, 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.MP3") //создание звукового объекта
Sound1.Play() //воспроизведение звукового объекта
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) //изменение громкости звукового объекта
Debug.trace("\n"+"Новая громкость звука: "+Sound1.GetVolume())
wait(10);Debug.trace("\n"+"Громкость устройства уменьшается")
SetVolume("Wave",5,10);wait(12) //изменение громкости звукового устройства
Debug.trace("\n"+"Новая громкость устройства: "+GetVolume("Wave"))
Debug.trace("\n"+"Финальная громкость звука: "+Sound1.GetVolume())
```

GetPosition

Позволяет получить текущую временную позицию воспроизведения звукового объекта.

Синтаксис:

GetPosition()

Возврат:

(Ч), текущая временная позиция воспроизведения указанного звукового объекта в секундах.

Пример:

```
var Sound1=OpenSound(SYSTEM_PUBLICATION_DIR+"MUSIC.MP3") //создание звукового объекта
Sound1.Play();wait(30) //воспроизведение звукового объекта
Debug.trace(Sound1.GetPosition()) //вывод в консоль позиции воспроизведения звукового объекта
```

SetPosition

Позволяет назначить нужную временную позицию воспроизведения звукового объекта. Можно использовать вместо функции перемотки.

Синтаксис:

SetPosition(From)

Параметры:

From - **(Ч)**, временная позиция в секундах, с которой будет воспроизводится звуковой объект. **ОП**.

Пример:

```
var Sound1=OpenSound(SYSTEM_PUBLICATION_DIR+"MUSIC.MP3") //создание звукового объекта
Sound1.SetPosition(30) //позиция старта воспроизведения устанавливается на 30 секунду
Sound1.Play() //воспроизведение звукового объекта с 30 секунды
```

QuickTime Virtual Reality (QTVR)

Функции **QTVR**-объектов позволяют управлять в публикации интерактивными панорамами со следующими поддерживаемыми форматами: **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.

Функции:

GetViewingLimits получить диапазоны углов и зума объекта **QTVR** (**не работает**).
GetPanAngle получить текущий панорамный угол объекта **QTVR** (в градусах).
SetPanAngle установить текущий панорамный угол объекта **QTVR** (в градусах).
GetTiltAngle получить текущий угол наклона объекта **QTVR** (в градусах).
SetTiltAngle установить текущий угол наклона объекта **QTVR** (в градусах).
GetFieldOfView получить текущий уровень масштабирования (зум) объекта **QTVR**.
SetFieldOfView установить текущий уровень масштабирования (зум) объекта **QTVR**.
GetCurrentNode получить идентификатор текущего узла объекта **QTVR**.
GetVisibleHotspots получить массив идентификаторов активных зон узла объекта **QTVR** (**не работает**).
GetHotspotType получить тип активной зоны текущего узла объекта **QTVR**.
EnableHotspot выборочно включить или отключить активные зоны объекта **QTVR**.
SetMouseOverCallback установить функцию, вызываемую при наведении курсора мыши на любую активную зону объекта **QTVR**.
SetHotspotCallback установить функцию, вызываемую при щелчке мыши на любой активной зоне объекта **QTVR**.

GetViewingLimits

Позволяет получить допустимые значения диапазонов углов и масштабирования указанного 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

Позволяет получить текущий панорамный угол (влево-вправо) указанного объекта **QTVR**.

Синтаксис:

GetPanAngle()

Возврат:

(Ч), текущий панорамный угол в градусах.

Пример:

```
Debug.trace("Панорамный угол: "+Panorama1.GetPanAngle())
```

SetPanAngle

Позволяет установить для указанного объекта **QTVR** 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

Позволяет получить текущий угол наклона (вверх-вниз) объекта **QTVR**.

Синтаксис:

GetTiltAngle()

Возврат:

(Ч), текущий угол наклона в градусах.

Пример:

```
Debug.trace("Угол наклона: "+Panorama1.GetTiltAngle())
```

SetTiltAngle

Позволяет установить для указанного объекта **QTVR** 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

Позволяет получить текущий уровень масштабирования (зума) указанного объекта **QTVR**.

Синтаксис:

GetFieldOfView()

Возврат:

(Ч), текущий уровень масштабирования (зум).

Пример:

```
Debug.trace("Уровень масштабирования: "+Panorama1.GetFieldOfView())
```

SetFieldOfView

Позволяет установить для указанного объекта **QTVR** 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

Позволяет получить идентификатор текущего узла в указанном объекте **QTVR**.

Синтаксис:

GetCurrentNode()

Возврат:

(Ц), идентификатор текущего узла (нумерация начинается с **1**).

Пример:

```
Debug.trace("Идентификатор текущего узла: "+Panorama1.GetCurrentNode())
```

GetVisibleHotspots

Позволяет получить массив идентификаторов активных зон заданного узла в указанном объекте **QTVR**.

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

Позволяет получить тип заданной активной зоны текущего узла в указанном объекте **QTVR**.

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

Позволяет выборочно включать или отключать активные зоны объекта **QTVR**. Функция работает **неправильно** и срабатывает на все активные зоны, независимо от введенных параметров.

Синтаксис:

EnableHotspot(Type,ID,Enable)

Параметры:

Type - **(Ц)**, активные зоны 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 возможное значение: **0**
Enable - **(Б)**, состояние работы активных зон (**true** - включить, **false** - отключить). **ОП**.

Пример:

```
Panorama1.EnableHotspot(3,0,false) //отключить все активные зоны объекта QTVR
```

SetMouseOverCallback

Позволяет установить функцию, вызываемую при наведении курсора мыши на любую активную зону объекта **QTVR**.

Синтаксис:

SetMouseOverCallback(MOHSCallback)

Параметры:

MOHSCallback - название пользовательской функции с одним параметром, вызываемой при наведении курсора мыши на любую активную зону объекта **QTVR**. **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:

```
//объявление функции, выполняемой при наведении курсора мыши на активную зону
function MouseOverQTVR(ObjHSMO)
{if (ObjHSMO.bEntering==true) {Debug.trace("Мышь над активной зоной № "+ObjHSMO.nID+"\n")}}
//назначение объявленной функции указанному объекту QTVR
Panorama1.SetMouseOverCallback(MouseOverQTVR)
```

SetHotspotCallback

Позволяет установить функцию, вызываемую при щелчке мыши на любой активной зоне объекта **QTVR**.

Синтаксис:

SetHotspotCallback(HSCallback)

Параметры:

HSCallback - название пользовательской функции с одним параметром, вызываемой при нажатии мышью на любую активную зону объекта **QTVR**. **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:

```
//объявление функции, выполняемой при нажатии мышью на активную зону
function MouseClickQTVR(ObjHSMC)
{Debug.trace("Идентификатор узла: "+ObjHSMC.nNodeID+"\n")
Debug.trace("Идентификатор активной зоны: "+ObjHSMC.nID+"\n")}
//назначение объявленной функции указанному объекту QTVR
Panorama1.SetHotspotCallback(MouseClickQTVR)
```

Поиск

Функции поиска позволяют находить страницы по ключевым словам (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 диалогового окна поиска и не нуждается в дополнительных функциях.

3) Функция **OpenSearch** и остальные предназначены для создания собственных диалоговых окон поиска.

Функции:

LaunchSearch отобразить встроенное диалоговое 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

Позволяет отобразить на экране встроенное диалоговое 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**.

Примеры:

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

OpenSearch

Позволяет создать объект поиска в публикации.

Синтаксис:

OpenSearch()

Возврат:

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

Пример:

```
var Search1=OpenSearch()
Debug.trace(Search1)
```

GetFirstKeyword

Позволяет получить первое ключевое слово в указанном объекте поиска.

Синтаксис:

GetFirstKeyword()

Возврат:

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

Пример:

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

GetNextKeyword

Позволяет получить следующее ключевое слово в указанном объекте поиска.

Синтаксис:

GetNextKeyword()

Возврат:

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

Пример:

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

GetFirstPage

Позволяет с помощью объекта поиска получить первую страницу, содержащую указанное ключевое слово.

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 Keyword=Search1.GetFirstKeyword()
var Page=Search.GetFirstPage(Keyword)
GotoPage(Page) //перейти на страницу
```

GetNextPage

Позволяет с помощью объекта поиска получить следующую страницу, содержащую указанное ключевое слово.

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 Keyword=Search1.GetFirstKeyword()
Search1.GetFirstPage(Keyword)
Search1.GetNextPage(Keyword)
GotoPage(Page) //перейти на страницу
```

Базы данных

Функции базы данных являются уникальными для объектов базы данных, созданных в скрипте. Следует создать новый объект базы данных в скрипте, который ссылается на **DSN**, прежде чем использовать функции базы данных.
Создание нового объекта базы данных:
Для файла **DSN** на диске **C**:
```
var DB1=new Database("FILEDSN=c:\catalogue.dsn;")
```
Для **DSN** с путем по умолчанию:
```
var DB1=new Database("FILEDSN=catalogue.dsn;")
```
Для машин **DSN**:
```
var DB1=new Database("DSN=catalogue;")
```

DATABASE_ERROR_MESSAGE - если использована база данных в публикации, эта переменная будет содержать любые сообщения об ошибках, отправленные через базу данных, если возникла проблема с подключением или доступом к базе данных.

Функции:
AutoCommit включить или выключить автофиксацию для базы данных.
Commit зафиксировать в базу данных - постоянно хранится в базе данных.
Connect подключиться к серверу **SQL**.
ExecuteSQL команда базы данных **SQL** для выполнения.
FirstRecord перейти к первой записи в наборе записей.
GetCurrentRecordNumber получить текущий номер записи в наборе записей.
GetIdentQuoteChar определить символ кавычки, необходимый для имен таблиц или полей, содержащих пробелы.
GetLastError получить последнее сообщение об ошибке базы данных.
GetNumberOfRecords получить общее количество записей в наборе записей.
GetRecordAt перейти к определенному номеру записи в наборе записей.
GetRecordAtRelative перейти к определенному номеру записи относительно текущего номера.
IsAutoCommitOn проверить, включен ли автофиксация в базе данных.
LastRecord перейти к последней записи в наборе записей.
NextRecord перейти к следующей записи в наборе записей.
PreviousRecord перейти к предыдущей записи в наборе записей.
Rollback вернуться к точке в базе данных до того, как были внесены изменения или последний отправленный коммит.

Базы данных - вступление

Пример показывает, в каком порядке следует использовать функции базы данных:

```
var DB1=new Database("DSN=Products;")
var RS1=DB1.ExecuteSQL("SELECT Price, Qty FROM Customer_Order WHERE OrderID=1;")
var total=0
//цикл установки для всех найденных записей
for (var i=1;i<=RS1.GetNumberOfRecords();i++) {
total+=RS1.Price*RS1.Qty
RS1.NextRecord()} //перейти к следующей записи
```

1. Создание нового объекта базы данных:

```
var DB1=new Database("DSN=Products;")
```

Новый объект базы данных подключается к базе данных с названием **Products**, которая содержит данные для работы. Он подключается к базе данных, используя протокол имени источника данных (**DSN**).

2. После того как установлено соединение с базой данных, следующая задача - получить скрипт для работы с точными данными, которые надо получить из базы данных. База данных просто хранит информацию в структурированном формате. Как правило, база данных состоит из нескольких таблиц, которые содержат записи. Каждая запись содержит поля, которые содержат информацию. В этом примере база данных называется **Products**, и она содержит таблицу **Customer_Order**, каждая запись в этой таблице содержит поля **Price**, **Qty** и **OrderID**. Язык под названием **SQL** (**Structured Query Language**) обычно используется для выбора точных данных, которые надо получить из базы данных. Все записи, соответствующие оператору **SQL**, сохраняются в наборе записей. Функция **ExecuteSQL** позволяет вводить оператор **SQL**, например:

```
var RS1=DB1.ExecuteSQL("SELECT Price, Qty FROM Customer_Order WHERE OrderID=1;")
```

Функция **ExecuteSQL** выберет поля **Price** и **Qty** из таблицы **Customer_Order**, если поле **OrderID** содержит значение **1**. Каждая запись, соответствующая этому оператору **SQL**, будет сохранена в переменной **RS1** - это набор записей для этого оператора **SQL**. Некоторые функции базы данных работают только с созданным набором записей, к ним относятся: **NextRecord**, **PreviousRecord**, **FirstRecord**, **LastRecord**, **GetRecordAt**, **GetRecordAtRelative**, **GetNumberOfRecords** и **GetCurrentRecordNumber**. Чтобы использовать эти функции, следует обратиться к переменной, содержащей название набора записей (в этом примере переменная называется **RS1**).

3. Теперь, когда создан набор записей, который соответствует нужным критериям выбора, можно использовать другие функции базы данных для выполнения множества других полезных операций. Функция цикла **for** используется с функцией базы данных **GetNumberOfRecords**:

```
for (var i=1;i<=RS1.GetNumberOfRecords();i++)
```

Функция **GetNumberOfRecords()** найдет число записей, содержащихся в переменной **RecSet** (новый набор записей, созданный этим сценарием). Он использует число записей в наборе записей для вычисления числа раз, когда он должен выполнять команды, содержащиеся в цикле.

4. Фактический цикл содержит две команды.

Первая команда это:

```
total+=RS1.Price*RS1.Qty
```

Эта команда выполняет вычисление и добавляет результат вычисления к переменной **total**. При расчете используется текущее значение, содержащееся в поле **Price** текущей выбранной записи в наборе записей, и умножает его на текущее значение, содержащееся в поле **Qty** выбранной записи в наборе записей.

Вторая команда это:

```
RS1.NextRecord()
```

Эта команда использует функцию базы данных **NextRecord** для перехода к следующей записи в наборе записей с именем **RS1**. Затем цикл запускается снова до тех пор, пока в наборе записей не останется больше записей.

ExecuteSQL

Позволяет выполнить допустимый оператор **SQL**, такой как команда **SELECT** или **INSERT INTO** для указанной базы данных.

Синтаксис:

ExecuteSQL(Command)

Возврат:

Для оператора **SELECT** возвращаемое значение - это новый объект набора записей; если есть ошибка, возвращаемое значение **false**.

Для оператора **UPDATE** или **INSERT** возвращаемое значение равно **true**, если действие работало правильно, и **false**, если оно не выполнялось.

Параметры:

Command - **(С)**, действительный оператор **SQL**. **ОП**.

Пример:

```
var DB1=new Database("FILEDSN=catalogue.dsn;")
var RS1=DB1.ExecuteSQL("SELECT * FROM Products")
```

Connect

Позволяет подключиться к серверу **SQL**. Нельзя подключиться к серверу **SQL**, если нет действительного идентификатора пользователя и пароля.

Синтаксис:

Connect(UserID,Password)

Возврат:

(Б), состояние подключения (**true -** соединение сделано, иначе **false**).

Параметры:

UserID - **(С)**, действительное имя ID для пользователя **SQL**. **НП**. Система может использовать значение по умолчанию, установленное на вкладке **Database** в диалоговом окне **Publication Properties**.

Password - **(С)**, действительный пароль для идентификатора пользователя. **НП**. Система может использовать значение по умолчанию, установленное на вкладке **Database** в диалоговом окне **Publication Properties**.

Примеры:

```
var DB1=new Database("DSN=catalogue.dsn;")
var dbConn=DB1.Connect("JSmith", password)
```

Commit

Позволяет 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("DSN=catalogue;")
DB1.AutoCommit(false)
if (DB1.ExecuteSQL("INSERT INTO Product (ProdName, Price) VALUES ("Product1", 100);")) {
if (DB1.ExecuteSQL("INSERT INTO Product (ProdName, Price) VALUES ("Product2", 200);"))
{DB1.Commit()} //сохранить изменения
else {DB1.Rollback()}} //отменить первую вставку, если вторая вставка идет плохо
```

RollBack

Позволяет вернуться к точке в базе данных до того, как были внесены изменения или была отправлена последняя фиксация. Для работы этой функции автофиксация должна быть выключена.

Синтаксис:

Rollback()

Пример:

```
DB1.Rollback()
```

AutoCommit

Позволяет включить или выключить функцию автофиксации, т.е. если **UPDATE** или **INSERT** данные, данные автоматически вводятся в базу данных.

Синтаксис:

AutoCommit(Enable)

Параметры:

Enable - **(Б)**, состояние автофиксации (**true** - включить, **false** - выключить). **НП**, по умолчанию **true**.

Пример:

```
var DB1=new Database("FILEDSN=catalogue.dsn;")
DB1.AutoCommit(false)
```

IsAutoCommitOn

Позволяет проверять, включена ли автофиксация.

Синтаксис:

IsAutoCommitOn()

Возврат:

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

Пример:

```
var DB1=new Database("DSN=catalogue;")
var status=DB1.IsAutoCommitOn()
```

GetLastError

Позволяет получить последнее отправленное сообщение об ошибке базы данных - если есть. Если выполнен оператор **ExecuteSQL** и он вернул ошибку, можно проверить отправленное сообщение об ошибке.

Синтаксис:

GetLastError()

Возврат:

(С), последнее сообщение об ошибке базы данных.

Пример:

```
var DB1=new Database("FILEDSN=catalogue.dsn;")
if (!DB1.ExecuteSQL("INSERT INTO Product (ID, Type) VALUES (10,'String')"))
{var ERRMSG=DB1.GetLastError()}
```

GetIdentQuoteChar

Позволяет получить символ кавычки, необходимый при цитировании имен таблиц базы данных и имен полей, в которых есть пробелы. Использовать его нужно только в том случае, если название таблицы или поля в базе данных содержит пробелы, например, если название таблицы - **New Products**, нужно знать символ кавычки, необходимый для правильной ссылки на оператор **SQL**.

Синтаксис:

GetIdentQuoteChar()

Возврат:

(С), символ кавычки.

Пример:

```
var DB1=new Database("DSN=catalogue;")
var qType=DB1.GetIdentQuoteChar()
DB1.ExecuteSQL("INSERT INTO"+qType+"New Products"+qType+"VALUES(10,'String')")
```

NextRecord

Позволяет получить следующую запись в наборе записей.

Синтаксис:

NextRecord()

Возврат:

Следующая запись в наборе записей если существует, иначе **false**.

Пример:

```
var DB1=new Database("FILEDSN=catalogue.dsn;")
var RS1=DB1.ExecuteSQL("SELECT * FROM Products")
RS1.NextRecord()
```

PreviousRecord

Позволяет получить предыдущую запись в наборе записей.

Синтаксис:

PreviousRecord()

Возврат:

Предыдущая запись в наборе записей если существует, иначе **false**.

Пример:

```
var DB1=new Database("FILEDSN=catalogue.dsn;")
var RS1=DB1.ExecuteSQL("SELECT * FROM Products")
RS1.PreviousRecord()
```

FirstRecord

Позволяет получить первую запись в наборе записей.

Синтаксис:

FirstRecord()

Возврат:

Первая запись в наборе записей если существует, иначе **false**.

Пример:

```
var DB1=new Database("DSN=catalogue;")
var RS1=DB1.ExecuteSQL("SELECT * FROM Products")
RS1.FirstRecord()
```

LastRecord

Позволяет получить последнюю запись в наборе записей.

Синтаксис:

LastRecord()

Возврат:

Последняя запись в наборе записей. Если записи отсутствуют, то **false**.

Пример:

```
var DB1=new Database("FILEDSN=catalogue.dsn;")
var RS1=DB1.ExecuteSQL("SELECT * FROM Products")
RS1.LastRecord()
```

GetRecordAt

Позволяет получить запись в запрошенной позиции в наборе записей.

Синтаксис:

GetRecordAt(Position)

Возврат:

Запись в указанной позиции в наборе записей. Если в этой позиции нет записи, то **false**.

Параметры:

Position - **(Ц)**, номер записи в наборе записей. **ОП**.

Пример:

```
var DB1=new Database("DSN=catalogue;")
var RS1=DB1.ExecuteSQL("SELECT * FROM Products")
RS1.GetRecordAt(3)
```

GetRecordAtRelative

Позволяет вернуть запись из набора записей относительно текущей позиции записи, т.е. если текущая позиция записи была **5** и относительная позиция была введена как **3**, восьмая запись в наборе записей будет возвращена.

Синтаксис:

GetRecordAtRelative(relativePosition)

Возврат:

Запись в указанной позиции относительно текущей позиции записи в наборе записей. Если в этой позиции нет записи, возвращаемое значение равно **false**.

Параметры:

relativePosition - **(Ц)** положительное или отрицательное, номер записи в наборе записей, к которому надо перейти, относительно текущей записи. **ОП**.

Пример:

```
var DB1=new Database("FILEDSN=catalogue.dsn;")
var RS1=DB1.ExecuteSQL("SELECT * FROM Products")
RS1.GetRecordAtRelative(-3) //3 записи до текущей записи
```

GetNumberOfRecords

Позволяет получить число записей в наборе записей. Если набор записей является подмножеством базы данных, возвращаемое значение для набора записей, а не базы данных.

Синтаксис:

GetNumberOfRecords()

Возврат:

(Ц), количество записей в наборе записей, НЕ в базе данных.

Пример:

```
var DB1=new Database("DSN=catalogue;")
var RS1=DB1.ExecuteSQL("SELECT * FROM Products")
var numTotal=RS1.GetNumberOfRecords()
```

GetCurrentRecordNumber

Позволяет получить номер текущей позиции записи в наборе записей. Если набор записей является подмножеством базы данных, возвращаемое значение для набора записей, а не базы данных.

Синтаксис:

GetCurrentRecordNumber()

Возврат:

(Ц), текущий номер позиции записи в наборе записей, НЕ в базе данных.

Пример:

```
var DB1=new Database("FILEDSN=catalogue.dsn;")
var RS1=DB1.ExecuteSQL("SELECT * FROM Products")
var recNum=RS1.GetCurrentRecordNumber()
```

Вопросы

Вопрос - особый тип объекта кадр с дополнительной вкладкой **"Question"** в свойствах.
GetQuestionCorrect получить правильный ли ответ.
GetQuestionIncorrect получить неправильный ли ответ.
GetQuestionPossible получить общее количество баллов.
GetQuestionValue получить сумму баллов.
SetUserAnswer указать ответ.
ResetUserAnswer сбросить ответ.
Confirm Question подтвердить ответ.
ResetQuestion сбросить вопрос.

Переменные вопросов

Q_SCORE_VALUE - значение или количество баллов пользователя до сих пор. Эта переменная отслеживает общую оценку ответов, которые пользователь получил правильно. Значение для отдельного ответа задается в сетке свойств вопроса. Это будет **8** в **8** из **10** до сих пор.
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 один.
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, независимо от того, набрал ли пользователь свои ответы. Это будет **10** в **8** из **10** до сих пор.
Q_SCORE_TOTAL_POSSIBLE - возможное значение всех вопросов во всей публикации. Это будет **10** в **8** из **10**, но для всей публикации.
Q_SCORE_VALUE_TOTAL_REMAINING - общая оценочная стоимость заданий в публикации, на которые еще не ответили.
Q_SCORE_CURRENTLY_POSSIBLE - отслеживает значение оценки, которую пользователь все еще может получить, если на все оставшиеся вопросы будут даны правильные ответы - включая текущее значение **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 текущий **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 всей викторине.
Q_SCORE_ANSWERED_QUESTIONS - **(Ц)**, число вопросов, которые пользователь пытался 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** - курс 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.

Q_TOPIC_ANSWERED_QUESTIONS - число ответов на вопросы.
Q_TOPIC_CORRECT - правильный счет для текущей темы.
Q_TOPIC_CURRENT_VALUE - значение пользовательских ответов для активной темы.
Q_TOPIC_CRITICAL_PASS - **false**, если пользователь решает критическое задание неправильно.
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 в теме.
Q_TOPIC_INCORRECT - неверный счет для текущей темы.
Q_TOPIC_NAME - **(C)**, название темы, установленное в свойствах публикации.
Q_TOPIC_PASSED - для проверки, был ли пройдена тема или нет.
Q_TOPIC_PASS_PERCENT - процент, необходимый для прохождения темы, как установлено в свойствах публикации.
Q_TOPIC_PASS_VALUE - фактические оценки, необходимые для прохождения темы, как установлено 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.
Они 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, которуюзадали в Свойствах публикации, сохраняется в переменной **TOPIC_VALUE_PERCENT[0]** и неправильные ответы, относящиеся ко второй теме, будут в **TOTAL_INCORRECT[1]**.

Скоринговая информация

SCORE_VALUE - вычисляет общую 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 - вычисляет общее значение распределенных баллов, на которые были даны ответы, независимо от того, правильно ли пользователь ответил на вопрос или нет. Используя приведенный выше пример (точка **1**), 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 - используйте это, чтобы датьчисло вопросов, на которые были даны правильные ответы, например, **5** из **7** вопросов.
SCORE_INCORRECT - используйте это, чтобы датьчисло вопросов, на которые были даны неправильные ответы, например, **2** из **7** вопросов.
SCORE_TOTAL - подсчитывает число вопросов, на которые пользователь попытался ответить, был ли этот ответ правильным или нет. Это не их общая оценка, которая хранится в **SCORE_VALUE**. Например, если есть **10** вопросов, пользователь попытался ответить на **7**, но получил только **4** правильных, общее количество баллов будет равно **7**.
SCORE_PERCENT - вычисляет процент вопросов, на которые были даны правильные ответы, например, 6 правильных ответов из **10** вопросов дают оценку **60%**, **6** из **15** вопросов - **40%**.
SCORE_TOTAL_POSSIBLE - определяемая пользователем переменная, в которой хранится максимально возможная оценка для всей викторины, независимо от того, сколько ответов ответил пользователь. Значение добавляется вручную.
SCORE_CURRENT_POSSIBLE - определяемая пользователем переменная для хранения общего возможного балла. Рассчитывается как **SCORE_TOTAL_POSSIBLE** меньше **SCORE_VALUE_TOTAL**
SCORE_PASS_THRESHOLD - определяемая пользователем переменная для хранения оценки, необходимой для достижения прохода. Это позволяет контролировать значение **SCORE_CURRENT_POSSIBLE** по отношению к отметке прохода и заставлять пользователя восстанавливать обучение, пока не будет поддержан уровень прохода.

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 запрошен специально или извлечен из имени пользователя с помощью функции **QuickScript**.
LOGIN_SURNAME - **(C)**, второе имя пользователя. Может быть запрошен специально или извлечен из имени пользователя с помощью функции **QuickScript**.
LOGIN_USER_EMAIL - **(C)**, адрес электронной почты пользователя. Для ввода пользователем/дизайнером.
LOGIN_TUTOR - репетитор курса. Для ввода пользователем/дизайнером.
LOGIN_MANAGER - 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**, сценарий создаст переменную объекта вопроса, которую могут видеть другие объекты, что было бы не так, если бы просто использовался **var** в самом сценарии. В качестве альтернативы можно создать переменную страницы для **UserScore** и возможного рейтинга, и тогда больше не понадобится это в любом из сегментов сценариев 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,проверяется, правильно ли на него был получен ответ, используя действие **if**, а затем отображается другой объект обратной связи - один для правильного ответа с названием **correctFeedback**, а другой - некорректный ответ

```
if (CorrectlyAnswered==true)
{correctFeedback.Show()}
else
{incorrectFeedback.Show()}
```

Наконец, может быть один объект с именем **feedbackPanel** с фрагментом текста, отображающим переменную **feedbackText**, и изменяющий **feedbackText** в зависимости от того, правильно ли ответил пользователь. Можно запустить фрагмент сценария, 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

Позволяет получить правильно ли пользователь ответил на конкретный вопрос. Будет нулевым, если пользователь ответил и (при необходимости) подтвердил ответ.

Синтаксис:

GetQuestionCorrect()

Возврат: **(Б)**, **true** или **false**.

GetQuestionIncorrect

Позволяет получить правильно ли пользователь ответил на конкретный вопрос. Будет нулевым, если пользователь ответил и (при необходимости) подтвердил ответ.

Синтаксис:

GetQuestionIncorrect()

Возврат: **(Б)**, **true** или **false**.

GetQuestionPossible

Позволяет получить общее количество баллов, доступных в данном вопросе пользователю при ответе на конкретный вопрос. Это **10** в **"8 из 10"**

Синтаксис:

GetQuestionPossible()

Возврат: **(Ц)**

GetQuestionValue

Позволяет получить сумму, набранную пользователем при ответе на конкретный вопрос. Это **8** в **"8 из 10"**. Будет нулевым, если пользователь ответил и (при необходимости) подтвердил ответ.

Синтаксис:

GetQuestionValue()

Возврат: **(Ц)**

SetUserAnswer

Позволяет указать ответ.

Синтаксис:

SetUserAnswer(AnswerIndex)

Параметры:

AnswerIndex - **(Ц)**, порядковый номер ответа, который надо установить через **Question Properties**. **ОП**.

ResetUserAnswer

Позволяет указать ответ, который будет сброшен для повторной попытки.

Синтаксис:

ResetUserAnswer(AnswerIndex)

Параметры:

AnswerIndex - **(Ц)**, порядковый номер ответа, который надо установить через **Question Properties**. **ОП**.

ConfirmQuestion

Позволяет подтвердить ответы, заданные для определенного вопроса, и дает **Opus** указание обновить оценку и другие переменные вопроса в соответствии с набором ответов и оценками, указанными в **Question Properties**. Отдельные ответы не могут быть изменены после этого действия, если предварительно не выполнено действие **ResetQuestion**.

Синтаксис:

ConfirmQuestion()

ResetQuestion

Позволяет сбросить все элементы вопроса и отменяет действие **Confirm Question**.

Синтаксис:

ResetQuestion()

Интернет и браузеры

Функции:
LaunchURL открыть **URL**.
InternetGetData получить информацию с удаленного сервера.
InternetPostData разместить информацию на удаленном сервере.
SendEmail отправить электронное письмо (требуется **Outlook**).
Navigate открыть указанный **URL** в указанном браузере.
Home перейти на домашнюю страницу указанного браузера.
Refresh повторно отобразить текущую страницу в указанном браузере.
Stop остановить указанный браузер.
Back вернуться назад на одну страницу в истории браузера.
Forward перейти вперед на одну страницу в истории браузера.
Print распечатать содержимое браузера.

LaunchURL

Позволяет запустить веб-браузер, содержащий указанный **URL** (унифицированный указатель ресурса).
Синтаксис:
LaunchURL(URL)
Возврат:
Указанный **URL**.
Параметры:
URL - **(С), URL** для запуска. **ОП**.
Пример:

```
LaunchURL("www.old-games.ru")
```

InternetGetData

Позволяет вернуть информацию с удаленного сервера.
Синтаксис:
InternetGetData(URL)
Возврат:
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

Позволяет отправлять и получать информацию с удаленного сервера.
Синтаксис:
InternetPostData(URL,Data,ReturnObject)
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 строковое значение). **НП**, по умолчанию **false**.
Пример:

```
var url="http://www.buka.com/cgi- bin/checkdata.pl" //настройка URL
var out=InternetPostData(url,"data=Opus") //вернет строку: "data=Opus"
out=InternetPostData(url,"data=Opus",true) //вернет объект с элементом данных data равным "Opus"
//Чтобы отправить объект, его сначала надо настроить
var obj=new Object()
obj.Name="John";obj.Age=21;obj.ShoeSize=8
out=InternetPostData(url,obj,true) //вернет объект с установленными свойствами Name, Age и ShoeSize
```

SendEmail

Позволяет отправить электронное письмо по указанному адресу электронной почты. Эта функция требует установленный **Outlook**.

Синтаксис:

SendEmail(ToList,CC,BCC,Subject,Message,Attachment,Showmail,MAPI)

Параметры:

ToList - **(С)**, адрес электронной почты получателя. **ОП**.

CC - **(С),** адрес электронной почты других получателей. **ОП**, но может иметь пустую запись, просто кавычки **""**.

BCC - **(С)**, адрес электронной почты других получателей, адрес этих получателей не будет виден получателю, названному в **ToList**. **ОП**, но может иметь пустую запись **""**.

Subject - **(С)**, тема заголовка 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 почтового диалогового окна перед отправкой почты (**true** - окно отображается, **false** - нет). **НП**, по умолчанию **true**.

MAPI - **(Б)**, отображение диалогового окна входа в систему электронной почты перед отправкой электронной почты (**true** - окно отображается, **false** - нет). **НП**, по умолчанию **false**.

Пример:

```
message.SetSelection(0,-1)
var emailText=message.GetSelectionText()
SendEmail("Jdoe@digitalworkshop.com","","","Welcome",emailText)
```

Navigate

Позволяет открыть запрошенный **URL** в указанном браузере.

Синтаксис:

Navigate(URL)

Параметры:

URL - **(С)**, **URL** страницы, которую надо открыть в указанном браузере. **ОП**.

Пример:

```
Browser1.Navigate("www.old-games.ru")
```

Home

Позволяет перейти на домашнюю страницу указанного браузера.

Синтаксис:

Home()

Пример:

```
Browser1.Home()
```

Refresh

Позволяет обновить текущую страницу, отображаемую в браузере.

Синтаксис:

Refresh()

Пример:

```
Browser1.Refresh()
```

Stop

Позволяет остановить указанный браузер.

Синтаксис:

Stop()

Пример:

```
Browser1.Stop()
```

Back

Позволяет перейти на одну страницу назад в истории указанного браузера.

Синтаксис:

Back()

Пример:

Browser1.Back()

Forward

Позволяет перейти на одну страницу вперед в истории указанного браузера.

Синтаксис:

Forward()

Пример:

Browser1.Forward()

Print

Позволяет распечатать содержимое Browser/DocView с помощью самой встроенной программы.

Синтаксис:

Print(ShowDialog)

Параметры:

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

Пример:

Browser1.Print(true)

Внешние DLL файлы

Внешние функции **DLL** относятся к объектам **DLL**, созданным в скрипте. Информация, возвращаемая из функции **DLL**, очевидно, может использоваться для других объектов. Функции **DLL** позволяют вызывать функции из внешних динамических библиотек (**DLL**) из **OpusScript**. Библиотеки **DLL** предоставляют способ выполнять более сложные операции с данными, чем это возможно в самом **OpusScript**. Информация, возвращаемая из функции **DLL**, может использоваться с другими объектами путем подключения соответствующих функций сценария для этих объектов к результату вызова **DLL**. Например, функции **DLL** могут вызывать **DLL**, чтобы позволить пользователю выбрать цвет из стандартного средства выбора цвета **Windows**, и затем можно поместить этот цвет в переменную, которая может быть использована для установки цвета многоугольника в публикации.

Функции:

LoadDLL загрузить **DLL**.

CallFn вызвать функцию из **DLL**.

Использование окон сообщений Windows

Функции для вызова внешних библиотек **DLL** позволяют получать доступ к объектам и функциям **Windows API**. Наиболее часто используемыми из них являются окна сообщений **Windows**, которые вызывают предупреждения или информацию с помощью кнопок **OK**, **Отмена** и других кнопок. Преимущество использования встроенных окон сообщений в том, что они соответствуют интерфейсу пользователя, а также быстрее, чем собственный дизайн.

Отображение окна сообщения

Окна сообщений **Windows** предоставляются в **DLL**-библиотеке **User32.DLL**, которая находится в системном каталоге для **Windows**. Его нельзя распространять вместе с публикацией, но он соответствует и требуется всем версиям **Windows**. Чтобы вызвать функцию, нужно использовать функцию **LoadDLL**, чтобы загрузить **DLL** в объект, а затем **CallFn**, чтобы вызвать функцию окна сообщения. При этом можно указать текст, который будет отображаться в окне сообщения и в строке заголовка окна, а также указать комбинацию значков и кнопок, которые будут использоваться. Окна сообщений окна содержат различные комбинации кнопок и значков (предупреждение, ошибка, информация, вопрос и т.д.), которые задаются путем предоставления единственного числа в качестве конечной части параметров, которые также можно использовать для установки кнопки по умолчанию. Комбинирует одно число из трех секций (их нужно объединять с помощью побитовых операторов, но в большинстве случаев это идентично простому сложению чисел).

Кнопки:

OK=0
OK и **CANCEL=1**
ABORT, RETRY и **IGNORE=2**
YES, **NO и CANCEL=3**
YES и **NO=4**
RETRY и **CANCEL=5**

Значки:

STOP=16
QUESTION=32
WARNING=48
INFO=64
Default Button:
First button=0
Second button=256
Third button=512
Fourth button=768

Доступна практически любая комбинация кнопок и значков, но ключевые:

Информационный значок с одной кнопкой **ОК** =64
Значок предупреждения с одной кнопкой **ОК** =48
Иконка Стоп с одной кнопкой **Отмена** =16
Предупреждение с кнопками **ОК** и **Отмена** =49
Вопрос с кнопками **Да** и **Нет** =36
Значок вопроса с кнопками **Да**, **Нет** и **Отмена** =35

Получение возвращенных значений

В некоторых случаях (когда в окне сообщений отображается более одной кнопки) необходимо узнать, какая кнопка в окне сообщений была нажата пользователем. Можно ответить соответственно. Возвращаемые значения для различных кнопок, доступных в диапазоне окон сообщений:

0 - ошибка
1 - нажата кнопка **"OK"**
2 - нажата кнопка **"CANCEL"**
3 - нажата кнопка **"ABORT"**
4 - нажата кнопка **"RETRY"**
5 - нажата кнопка **"IGNORE"**
6 - нажата кнопка **"YES"**
7 - нажата кнопка **"NO"**
10 - нажата кнопка **"TRY AGAIN"**
11 - нажата кнопка **"CONTINUE"**

В примере пользователю предоставляется возможность подтвердить выход из публикации. Отображается значок вопроса с кнопкой **"YES"** или **"NO"**, а затем выходит из публикации, только если была нажата кнопка **"YES"** (возвращая значение **6** в переменную **UserAnswer**).

```
var UserDLL=LoadDLL(SYSTEM_WINSYS_DIR+"User32.dll")
if (UserDLL) {
var UserAnswer=UserDLL.CallFn("MessageBoxA",true,"slong","hwnd",0,"string","Are you sure you want to exit the publication?","string","Confirm Exit","ulong",36)
if (UserAnswer==6) {ExitPublication()}}
```

LoadDLL

Позволяет загрузить **DLL**, чтобы позволить функциям из внешних динамических библиотек вызываться из скрипта. **DLL** предоставляют способ выполнять более сложные операции с данными. **DLL** должна экспортировать функции как стандартные недекорированные функции соглашения о вызовах языка **Си**.

Синтаксис:

LoadDLL(Filename)

Возврат:

(O), для доступа к функциям в **DLL**.

var MyDLL=LoadDLL("MyFunctions.dll")

Сделает функции, содержащиеся в **MyFunctions.dll**, в объекте **MyDLL**, чтобы к ним можно было получить доступ, используя, например, **CallFn** как функцию этого объекта:

MyDLL.CallFn(Function,HasDisplay,Return,params)

Если **DLL** не может быть загружена, возвращаемое значение равно нулю. Поэтому рекомендуется проверить, что библиотека **DLL** открыта перед вызовом ее функций, поместив вызовы функций в оператор **if** для библиотеки **DLL** с названием **MyDLL**:

if (MyDLL) {function calls}

Параметры:

Filename - **(C)**, путь к загружаемому **DLL**-файлу. **ОП**.

Пример:

Загрузка системной библиотеки **Windows User32.dll**. Функция отображает стандартное окно сообщения **Windows**. Стиль окна сообщения является одним из нескольких встроенных стилей, предоставляемых **User32**, и определяется длинным целым числом **48**.

```
var UserDLL=LoadDLL(SYSTEM_WINSYS_DIR+"User32.dll")
if (UserDLL) {
UserDLL.CallFn("MessageBoxA",true,"slong","long",0,"string","This is a message box!","string","This is the title","long",48)}
```

CallFn

Позволяет вызывать функции из внешних **DLL**. Любая переменная может рассматриваться как любой тип и будет автоматически преобразована без предупреждения. Если для **DLL** требуются более сложные типы аргументов, чем те, которые поддерживаются **CallFn**, возможно, можно написать **DLL**-wrapper, которая содержит упрощенный интерфейс, который использует только типы данных, понятные **ecmascript**.

Синтаксис:

CallFn(Function,HasDisplay,Return,params <Type,Value,Type,Value…>)

Названия функций и представления типов заключаются в кавычки, а фактические значения (числа, **true**, **false**) - нет. Шевроны, заключающие в скобки ряд параметров, приведены для ясности и не должны быть включены.

Возврат:

Значение типа, указанного в параметрах вызова. В случае любой ошибки возвращаемое значение равно **-1**.

Параметры:

Function - **(С)**, название вызываемой функции, должно совпадать с экспортируемой функцией в **DLL**. **ОП**.

HasDisplay - **(Б)**, будет ли вызов **DLL** отображать пользовательский интерфейс, и, таким образом, интерфейс будет активным. **ОП**.

Return - **(С)**, тип возвращаемого значения из функции (если есть), возможные значения:

"none" void
"uchar" unsigned char
"schar" signed char
"ushort" unsigned short
"sshort" signed short
"ulong" unsigned long
"slong"signed long
"float" float
"string" unsigned char pointer (unsigned **char***) отвечает за то, чтобы указанная память оставалась действительной после возврата вызова функции.

Если возвращаемое значение не требуется, но функция **DLL** объявляет его, то следует указать его тип для правильного вызова функции. **ecmascript** 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 - первый - это тип параметра, а второй - значение для передачи: целые числа (**unsigned long**) или строки с нуль-терминатором в конце (**char ***).

Включен специальный тип, **hwnd**, который обеспечивает дескриптор окна, в котором происходит вызов (при необходимости). Позволяет авторам **DLL** создавать свои дисплеи как дочерние элементы окна публикации и, таким образом, например центрировать свое отображение в окне публикации. Значение в паре тип/значение игнорируется, поэтому рекомендуется просто установить **0**.

"ulong" unsigned long
"slong" signed long
"float" float
"string" unsigned **char***
"hwnd" прикрепление к окну.

Пример:

```
var TestDLL=LoadDLL(SYSTEM_PUBLICATION_DIR+"CustomDLL.dll") //загрузка внешней DLL
//Вызов функции Action без возвращаемого значения ("none") и без параметров (не указываются)
if (TestDLL) {TestDLL.CallFn("Action",false,"none")}
```

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.