Roboocyte2 JavaScript Documentation

Roboocyte2 Version 1.1.5 2011-11-03

Roboocyte2 scripts

Scripts are written in the Roboocyte2 text editor. They are saved in ASCII format and thus can be edited also using a standard text editor (like Notepad). It must NOT be opened with a text processing program like Microsoft Word or Open/Libre Office Writer.

Summary of the Roboocyte2 JavaScript syntax

- JavaScript is case sensitive
- only one statement should be written in one line
- statements should be terminated with a semicolon (';')
- statements can be grouped in code blocks within curly brackets `{}`
- a commented line starts with `//` (everything after // is not executed)
- multiline comments start with `/*` and end with `*/`
- variable names must begin with a letter or an underscore, numbers are allowed within a name, no other characters, especially no blanks are allowed. Reserved words cannot be used as variable names
- numbers are either integers (e.g. `420`) or floating point numbers (e.g. `3.1`). The decimal symbol must be a dot.
- a string is text within double or single quotes (`„text2“`, `'text2'`)
- boolean values are `true` and `false`

Brief documentation

Here we describe only the most important JavaScript elements, see Further Reading for more detailed descriptions and references.

For a working example with the Roboocyte2 software see the examples scripts, especially `JavaScript_Example.js`

Data types

A value, the data assigned to a variable, may consist of any sort of data. However, JavaScript considers data to fall into several possible *types*. Depending on the type of data, certain operations may or may not be able to be performed on the values. For example, you cannot arithmetically multiply two string values. Variables can be these types[1]:

Type	Description	Example
Integer number	integral numbers, negative numbers are prefixed by -	`1`, `-3`, `0`
floating point number	decimal point must be a dot: `'.'`	`3.14`,
string	text within single or double quotes	`„Hello“`

1 quoted from: http://www.wdvl.com/Style/JavaScript/Tutorial/variables.html

`Undefined`	the value of a variable before it gets a value by assignment. Occurs in an array when not all members have been assigned	

Arithmetic operators

Mathematical calculations on numbers are made by using the followeing arithmetic operators:

Operator	**Description**	**Example**
`+`	addition	`y = x + 2;`
`-`	subtraction	`y = x - 2;`
`*`	multiplication	`y = x * 2;`
`/`	division	`y = x / 2;`
`++`	increment by 1, usually used in a `for` loop	`i++`
`--`	decrement by 1	`i--`

String functions

Strings can be manipulated by a number of functions operating on them. The addition operator „+“ has the meaning of concatenation when used on strings:

Function	**Paramters**	**Description**	**Example**
`+ operator`	-	concatenation	`'up' + 'date'` yields `'update'`
`length`	-	number of characters in the string	`var s3 = 'update';` `s3.length` is 6
`[index]`	-	get the character with the given index, starting at 0	`s3[2]` is 'd'
`substring`	int: startindex, int: stopindex	returns the substring from startindex to stopindex	`s3.substring(1, 3)` is 'pd'
`split`	string: separator	splits string in a part for each occurence of separator, returns an aray of the string parts	var s4 = "a:b:cde:fg:h"; gives an array
`indexOf`	string: substring	returns the index of the first occurence of the substring, -1 if not found	
`concat`	list of strings	concatenates the string with all strings in the parameter list	

Variables and assignment operators

Variables are used to store values. The value can be retrieved later in the script by using the variable name. A new variable should be declared with the keyword `var`.

A value is assigned to a name by using an assignment operator (`=`, `+=`, `-=`, `*=`, `/=`). Usually `=` is used, e.g.

```
var x = 50;
```

or

```
var minimumResult = -50;
```

The other operators are shortcuts for:

`x += 2;` is equal to `x = x + 2;`

`x -= 2;` is equal to `x = x - 2;`

`x *= 2;` is equal to `x = x * 2;`

`x /= 2;` is equal to `x = x / 2;`

Conditions and comparison operators

Comparison operators are normally used with conditional statements. By using these (with keywords **`if`** and **`else`**) you can execute different parts of the script depending on certain conditions. A condition statement is started with the keyword **`if`**, followed by a condition expression within parenthesis and statements within a block (within {}). It can optionally be followed by an **`else`** statement.

```
if (condition)
{
        statements1
}
```

or

```
if (condition)
{
        statements1
}
else
{
        statements2
}
```

Conditional statements can also be nested:

```
if (condition1)
{
      <optional statements>
      if (condition2)
      {
        statements
      }
}
```

The condition is usually a comparison expression. Comparison operators are

Operator	**Description**	**Example**
`==`	equal	`x == 2`
`!=`	not equal	`x != 2`
`>`	greater	`x > 2`

`>=`	greater or equal	`x >= 2`
`<`	less	`x < 2`
`<=`	less or equal	`x <= 2`

Loops

With loops it is possible to repeatedly execute parts of the script. There are two kinds of loops:

1. For loop: This is normally used if the number of repetitions is fixed or known in a variable. The for loop is started with the keyword **`for`**, followed by three expressions within parenthesis and a statement block within **`{}`**:
   ```
   for (initial-expression; condition; increment-expression)
   {
     statements
   }
   ```
   The **`initial-expression`** initializes the counting variable, the **`condition`** defines when the loop is to be finished, the **`increment-expression`** is used to increment the counting variable in each cycle of the loop.
2. While loop: The number of repetitions need not be known at the time of writing the script, the loop is terminated by a condition which is usually calculated at the time when the script is executed only. The syntax is similar to an **`if`** statement: The keyword **`while`** is followed by a condition and a statement block within **`{}`**:
   ```
   while (condition)
   {
    statements
   }
   ```

With the keyword **`break`** the loops can be terminated at any time. When **`break`** is excuted, the loop is aborted and script execution continues after the loop code block.

With the keyword **`continue`**, the code inside a loop can be skipped. When **`continue`** is excuted, the script immediately continues

Arrays

Arrays are used to store several values in one variable. There are two ways to create an array variable:

1. via the new keyword: **`var TestArray = new Array();`** or **`var TestArray = new Array(1, 2, 3, 4);`**
2. via square brackets: **`var TestArray2 = [1, 2, 3, 4];`**

The values are stored in the array by using an index, starting with 0 for the first entry in the array. The length of an array is determinded dynamically: It is the index + 1 of the last defined value in the array. Values in the arra can be of all available data types: numbers, strings, even other arrays

Several operations are possible on arrays:

- **`length`** property: get the number of elements in the array
- square brackets **`[]`**: set or retrieve values by index
- **`toString()`**: gets a text representation of all values
- **`push(value)`**: add value to the end of the array
- **`pop()`**: retrieve and remove last value

- `shift()`: move all values to their previous position, thereby removing the first element and shortening the length by 1
- `join(connecting string)`: connects the values of the array with the argument and returns a string

Functions

Functions are a very powerful concept in JavaScript. Here only the most basic features are documented.

Functions can be used to hold code, which is used for a specific purpose, in a separate block. This make it much easier to maintain a clear structure in the script code, especially when the function is used in several different places.

Functions are defined via the keyword function, followed by the user defined function name and a code block. The function definition also specifies parameters. These are names for values which are used when the function code is executed. The code of the function is executed by calling it with the given name and the concrete values of the parameters.

If a variable is declared with the keyword `var` inside the function code block, it is local to the function and can not be referenced from other code.

Example function definition:

```
function square(x) {
      return x * x;
}
```

To execute the code with actual parameter values:

```
var squared = square(2);
```

Further Reading

There are many tutorials and references about JavaScript to be found on the internet, a few of them are listed below. Remember that only the JavaScript syntax is used, all web specific elements are not supported, so in the tutorials all references to HTML, especially tags and the **document** object, to the **DOM** model, to events and to web pages must be ignored.

http://www.w3schools.com/js

http://www.echoecho.com/javascript.htm (see the JAVASCRIPT BASICS)

http://www.cs.brown.edu/courses/bridge/1998/res/javascript/javascript-tutorial.html

http://www.webmonkey.com/2010/02/javascript_tutorial/

http://www.wdvl.com/Style/JavaScript/Tutorial/index.html