synthiam.com

Migrating from EZ-Script to JavaScript

How to move your code from EZ-Script to JavaScript. JavaScript offers faster execution and more features.

Last Updated: 2/2/2022

Why?

Why migrate from EZ-Script to JavaScript? The answer is performance and functionality. While EZ-Script appears straightforward, it is very similar to JavaScript. The most significant differences will be outlined in this tutorial. However, JavaScript can organize commands in groups, called classes. These command groups make JavaScript more organized and easier to find the correct command that you're looking for.

Performance

The Synthiam implementation of JavaScript in ARC is fast... very fast! It outperforms EZ-Script by a significant magnitude. Take for instance, a benchmark between the two languages by counting to 1,000,000.
JavaScript (1.1 seconds)

EZ-Script (2 minutes, 5 seconds)

Organized Commands

JavaScript organizes commands in groups, called classes. There are classes for controlling servos, reading ADC, setting and reading digital ports, and more. The methods within the classes are well documented on the JavaScript manual page in the support section.
ARC JavaScript Manual: https://synthiam.com/Support/javascript-api/javascript-overview

How Does It Work

When typing in the JavaScript window, intellisense will detect the first characters of each word and display a dropdown of options. For example, if want to move a servo into a position, we begin typing
Servo
. and the class for Servo commands will display.

Each class separates the commands by a (.) period. Select the class and the commands within that class will be displayed.

And here we can see the command Servo.setServoPosition, to move a servo into a specified position. There are variants of the command that support different parameters. That means you can type any variant of that command by providing only the parameters that you wish to provide. For Servo.setServoPosition, there is one variant that accepts the board index, and the other does not which means it defaults to index #0
Either of these two servo commands will perform the same outcome. The only difference is the second command is specifying the EZB board index.

```
Servo.setPosition(d0, 75);
Servo.setPosition(d0, 75, 0);
```

Semicolon Line Endings

When writing EZ-Script code, at the end of the line you simply press <enter> and start coding on the next line. However, in JavaScript, it is recommended to terminate the line with a semicolon ; character. With the ARC JavaScript compiler, it is not mandatory to do it, but you will find most code examples will have the semicolon at the end of the line.

Examples

EZ-Script Example

```
print("This is some text")
print("This is another line of text")
```

JavaScript Example

```
print("This is some text");
print("This is another line of text");
```

*Note: Notice the semicolon at the end of the lines in JavaScript. Again, it's not mandatory, but it is recommended for proper JavaScript compliance.

Why Semicolons?

Traditionally, a semicolon has been common across many programming languages to tell the compiler the end of the command. This is because many commands can be included on the same line with JavaScript. It is not recommended to do this because it is messy to read, but some people program this way.

Example Of Many Commands On One Line

```
print("Hello"); print("World");
```



Case Sensitive

JavaScript, much like many programming languages, is case sensitive for variabes and functions. That means a variable named "MyVariable" is different than "myVariable". This applies to all variables, commands and functions.

Example

Here is an example program that demonstrates how the different case of a variable makes two different variables.

```
var MyVariable = "ONE";
var myVariable = "TWO";
print(MyVariable);
print(myVariable);
```

The output of that example will demonstrate how ONE and TWO are both printed. This is because the case sensitive variable name is taken into consideration for that variable.

Variables

The first thing to cover are how variables differ. A variable in EZ-Script starts with a $ (dollar sign). A variable in JavaScript does not start with anything and can be named however you wish, as long as the first character is not a number.

Assign Variable Example

EZ-Script Example
Here is an example of a variable being created in EZ-Script

```
$MyVariable = "Some text"
print($MyVariable)
```

JavaScript Equivalent
This is the equivalent of assigning a value to a JavaScript variable

```
var MyVariable = "Some text";
print(MyVariable);
```

*Note: Remember that variables are case-sensitive. This is covered in an previous step of this tutorial.

JavaScript Variables Are Not Global Across ARC

This is one difference that you must be aware of. When creating a variable in EZ-Script, the variable is accessible across all robot skills in ARC. However, when creating a variable in JavaScript, it is only accessible by that isolated script. To share variables across other robot skills, you will use the
getVar()
and
setVar()
commands. That will set a global variable with a value so other robot skills can see it.
In fact, setting and getting variables from the Global Variable storage will also work across EZ-Script variables. EZ-Script sets variables as global by default. So, if you assign a variable in EZ-Script, you can retrieve it in another robot skill using JavaScript with the *getVar()* command.

Example of Using Global Variables
If you have a variable that you would like to share with other robot skills, use the s*etVar()* function. This will push the variable's current value into the global stack so it is available by other robot skills.

```
var myVariable = "This is some text";
setVar("$myVariable", myVariable);
```

In that example, the global $myVariable will be assigned the value of JavaScript value of myVariable. This means that in another robot skill, you can retrieve that value.

```
```

```
var myVariable = getVar("$myVariable");
print(myVariable);
```

[head2]Global Variable Viewer[/head]
The global variable viewer will display all variables in ARC. By clicking on a variable, it will insert the *setVar()* command where the cursor is.

JavaScript = uses ==

That's right, when comparing two values in JavaScript IF condition, use == instead of a single =. This is because a single = is reserved for assigning values. The double == is reserved for comparing values.

```
var myVariable = 5;
if (myVariable == 5) {
print("You got it");
}
```

That's the only difference for comparisons. Using > greater than, < less than, >= greater than equal to, <= less than equal to, etc.. are all the same.
[head2]Not Equals[/head]
Not equals puts an *!* exclamation mark in front of the = equal sign in the comparison. This means that the value is not equal to another value.

```
var myVariable = 5;
if (myVariable != 10) {
print("The value does not equal 10");
}
```

JavaScript Uses Brackets

In EZ-Script, an IF condition ends with an ENDIF. JavaScript operates the same way but uses instead. This is called the scope, when code is between the brackets. In EZ-Script, the scope is between the IF and ENDIF statements. In JavaScript, the scope is between the closing bracket.
EZ-Script

```
$myVariable = 5
if ($myVariable == 5)
print("You got it")
endif
```

JavaScript

```
var myVariable = 5;
if (myVariable == 5) {
print("You got it");
}
```

IF ELSEIF ELSE

This is pretty much the same as EZ-Script, by having ELSE IF and ELSE commands.

```

var myVariable = 5;
if (myVariable == 5) {
  print("You got it");
} else {
  print("Nope, missed");
}

```

or similar example using ELSE IF

```

var myVariable = 5;
if (myVariable == 5) {
  print("You got it");
} else if (myVariable = 6) {
  print("So close");
} else {
  print("Nope, missed");
}

```

Single Command Doesn't Require Brackets

Occasionally you may see an IF condition with no brackets. That is because you may only have one command after the IF, therefore you do not require brackets. This is similar to the basic programming language, where a single IF that doesn't have ENDIF means only use the next command.

EZ-Script

```
$myVariable = 5
if ($myVariable = 5)
print("You got it")
endif
```

JavaScript

```
var myVariable = 5;
if (myVariable == 5)
print("You got it");
```

For Loop

EZ-Script has a few different methods to loop. You can loop with EZ-Script using the REPEAT, REPEATUNTIL, REPEATWHILE, and simple GOTO. We'll cover the equivalent JavaScript commands here.

REPEAT = FOR

The EZ-Script REPEAT command is equivalent to the FOR command in JavaScript.

EZ-Script

```
REPEAT($cnt, 1, 10, 1)
print($cnt)
ENDREPEAT
```

JavaScript

```
for (var cnt = 0; cnt < 10; cnt++) {
  print(cnt)
}
```

Here is an explanation of each command in the FOR statement from the W3 Schools website.

Quote:
for (
statement 1
;
statement 2
;
statement 3
) {
//
code block to be executed
}
Statement 1
is executed (one time) before executing the code block.
Statement 2
defines the condition for executing the code block.
Statement 3
is executed (every time) after the code block has been executed.

REPEATWHILE = WHILE

The EZ-Script REPEATWHILE() command is replaced with WHILE() in JavaScript. They operate the same way and accept the same parameter. The loop will continue while the condition is TRUE.

EZ-Script

```
$cnt = 0
repeatwhile( $cnt < 5)
```

```
  print($cnt)

  $cnt = $cnt + 1
endrepeatwhile
```

JavaScript

```
var cnt = 0;
while (cnt < 5) {
  print(cnt);

  cnt = cnt + 1;
}
```