

# TypeScript Implementation in Aria Automation

**BWI**  
IT für Deutschland





# Agenda

01

Introduction

02

TypeScript and  
Security Benefits

03

Implementation

04

Updates and Runtime  
Environment Updates

05

Findings, Miscellaneous and  
Questions



# Introduction

- Stefan Schnell
- 60 Years
- Nearly four years at BWI GmbH as Senior IT-Architect at Cloud & Platform Technologies department
- Focus on data center automation  
with VMware Aria Automation (formerly vRealize Automation – vRA)
- Specialized on integration scenarios with different platforms, interfaces and programming languages
- More information:  
<https://stschnell.de/> - Private homepage  
<https://blog.stschnell.de/> - VCF Automation blog  
<https://linkedin.com/in/stefan-schnell/> - LinkedIn profile



# TypeScript

- TypeScript is a scripting language developed by Microsoft. It bases on the ECMAScript 6 standard.
- It was developed by Microsoft to simplify the development of large JavaScript applications.
- It is a superset of JavaScript, because it transpiles the TypeScript source code into JavaScript code. This means that every JavaScript program is a TypeScript program.
- TypeScript adds static typing to JavaScript. This allows to add a specific type to each variable, e.g. String, Boolean or Number. But TypeScript offers by Design no guarantees for type safety.
- Furthermore the use of extended functions such as classes (OOP), inheritance, modules, etc. is possible.
- TypeScript is an advantage when working in larger projects with several developers.
- It can help to build a consistent code base and avoid errors that can occur when different developers use different JavaScript conventions.

# Security Benefits

- The TypeScript transpiler finds type errors during transpilation (compilation time). It prevents that these errors are passed to the runtime.
- This static typing prevents problems before they can become security vulnerabilities.
- TypeScript helps with validation and maintaining of the type safety.

# TypeScript / JavaScript

```
function addNumbers(a: number, b: number) : number {  
    const retValue: number = a + b;  
    return retValue;  
}  
const sum: number = addNumbers(10, 15);  
const sumErr: number = addNumbers("Stefan", 16);  
// Argument of type 'string' is not assignable to parameter of type 'number'
```

---

```
function addNumbers(a, b) {  
    var retValue = a + b;  
    return retValue;  
}  
var sum = addNumbers(10, 15);  
var sumErr = addNumbers("Stefan", 16);
```

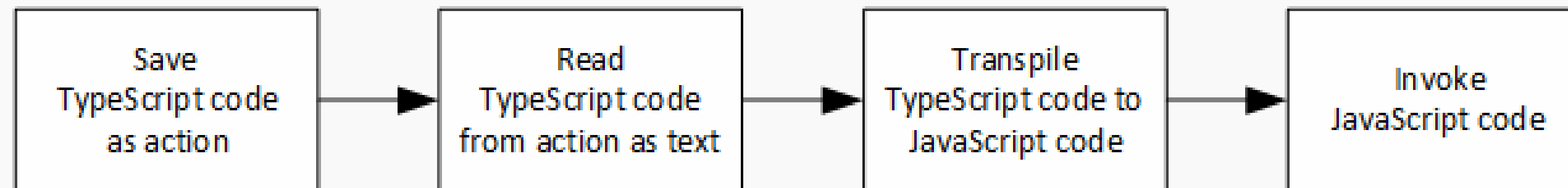
# Implementation (1) – Requirements

- The TypeScript transpiler from Microsoft has to be implemented.
- It has to be implemented on such a way that the storage of the TypeScript codes, the transpilation and the execution of the resulting JavaScript code is completely in Aria Automation.

# Implementation (2)

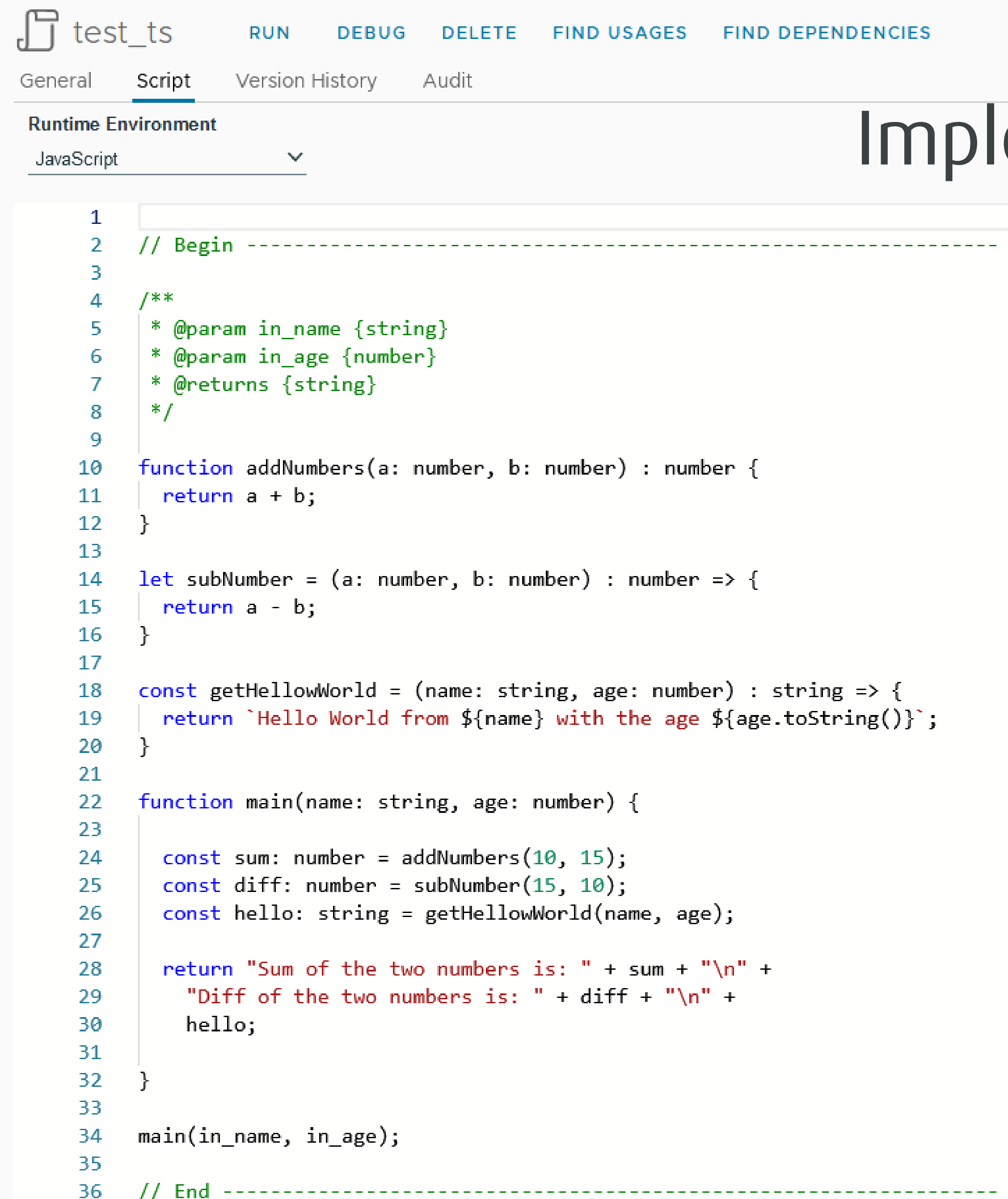
## Native TypeScript-Code

- The TypeScript code is saved in an action and read with `getActionAsText`.
- The TypeScript code is then transpiled, with the JavaScript action `transpileTypeScript`, which calls the Node Action `transpileTypeScriptToJavaScript`.
- Finally the generated JavaScript code is executed with `invokeTypeScript` (`evaluateString`).



- More informationen about `evaluateString`:  
<https://www.javadoc.io/doc/org.mozilla/rhino/1.7R4/org/mozilla/javascript/package-summary.html>







```
1
2 // Begin -----
3
4 /**
5  * @param in_name {string}
6  * @param in_age {number}
7  * @returns {string}
8  */
9
10 function addNumbers(a: number, b: number) : number {
11     return a + b;
12 }
13
14 let subNumber = (a: number, b: number) : number => {
15     return a - b;
16 }
17
18 const getHelloWorld = (name: string, age: number) : string => {
19     return `Hello World from ${name} with the age ${age.toString()}`;
20 }
21
22 function main(name: string, age: number) {
23
24     const sum: number = addNumbers(10, 15);
25     const diff: number = subNumber(15, 10);
26     const hello: string = getHelloWorld(name, age);
27
28     return "Sum of the two numbers is: " + sum + "\n" +
29         "Diff of the two numbers is: " + diff + "\n" +
30         hello;
31 }
32
33
34 main(in_name, in_age);
35
36 // End -----
```

# Implementation (3)

# Implementation (4)

 testTypeScriptExample RUN DEBUG DELETE FIND USAGES FIND DEPENDENCIES Completed

General Script Version History Audit

**Runtime Environment**  
JavaScript 

```
1
2 // Begin -----
3
4 var result = System.getModule("de.stschnell.library.typeScript")
5   .invokeTypeScript(
6     "de.stschnell.library.typeScript",
7     "test_ts",
8     "{\"in_name\": \"\\\\\"Stefan\\\\\"\", \"in_age\": 42}",
9     false
10  );
11
12 System.log(result);
13
14 // End -----
15
```

Result / Inputs Logs

```
2024-09-03 14:29:51.353 +02:00 INFO (de.stschnell/testTypeScriptExample) Sum of the two numbers is: 25
Diff of the two numbers is: 5
Hello World from Stefan with the age 42
```



# Implementation (5)

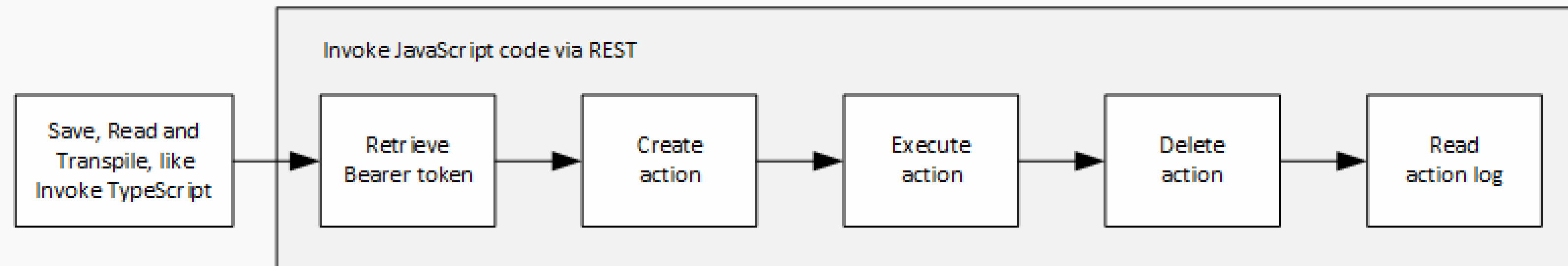
## Using Aria Automation Objects

- The use of Aria Automation objects requires a different approach.
- The use of native TypeScript takes place in a separate context.  
This means that the executed JavaScript code loses its connection to the caller.
- This can be avoided by using the Aria Automation Orchestrator API.
- This makes it possible to use the entire scope of Aria Automation objects.

# Implementation (6)

## TypeScript-Code mit Aria Automation Objekten

- The execution of save, read and transpile is the same as for native TypeScript code.
- But the generated JavaScript code is executed with the JavaScript action `invokeMixScript`.





# Implementation (7)

```

15 interface Point {
16   x: number;
17   y: number;
18 }
19
20
21 function logPoint(p: Point) {
22   System.log(`${p.x}, ${p.y}`);
23 }
24
25 enum CardinalDirections {
26   North,
27   East,
28   South,
29   West
30 }
31
32 class Person {
33   private name: string;
34
35   public constructor(name: string) {
36     this.name = name;
37   }
38
39   public getName(): string {
40     return this.name;
41   }
42 }
43
44 function getVcenters() {
45
46   const vcenters: Array<string> = VcPlugin.allRegisteredInstances;
47
48   vcenters.forEach( (vcenter) => {
49     System.log(vcenter);
50   });
51 }
52
53

```

```

53
54 function main() {
55
56   const sum: number = addNumbers(10, 15);
57   const diff: number = subNumber(15, 10);
58   const prod: number = mulNumber(4, 4);
59
60   System.log("Sum of the two numbers is: " + sum);
61   System.warn("Diff of the two numbers is: " + diff);
62   System.error("Product of the two numbers is: " + prod);
63
64   const point = { x: 12, y: 26 };
65   logPoint(point);
66
67   let multilineString: string = `
68     This is a multiline string.
69     In TypeScript, we use backticks.
70     It makes the code more readable.
71   `;
72
73   System.log(multilineString)
74
75   let currentDirection = CardinalDirections.North;
76   System.log("Current Direction is: " + currentDirection);
77
78   const person = new Person("Stefan");
79   System.log(person.getName());
80
81   getVcenters();
82
83 }
84
85 main();
86

```

## Runtime Environment

JavaScript 

## Inputs:

 in\_userName : string
  in\_passWord : SecureString

```

1  |
2  // Begin -----
3
4  var moduleName = "de.stschnell.library.typeScript";
5  var actionName = "testMix_ts";
6
7  var result = System.getModule("de.stschnell.library.typeScript").invokeMixScript(
8      in_userName,
9      in_passWord,
10     moduleName,
11     actionName
12 ).actionLog;
13
14 var jsonResult = JSON.parse(result).logs;
15
16 Object.keys(jsonResult).forEach( function(item) {
17     System.log(jsonResult[item].entry["short-description"]);
18 });
19
20 // End -----

```

Result / Inputs Logs

```

2024-09-03 14:49:31.794 +02:00 INFO (de.stschnell/testMixTypeScriptExample) (temp/x7557e207_64d0_4f84_9670_08b74f9d259d) Sum of the two numbers is: 25
2024-09-03 14:49:31.795 +02:00 INFO (de.stschnell/testMixTypeScriptExample) (temp/x7557e207_64d0_4f84_9670_08b74f9d259d) Diff of the two numbers is: 5
2024-09-03 14:49:31.796 +02:00 INFO (de.stschnell/testMixTypeScriptExample) (temp/x7557e207_64d0_4f84_9670_08b74f9d259d) Product of the two numbers is: 16
2024-09-03 14:49:31.797 +02:00 INFO (de.stschnell/testMixTypeScriptExample) (temp/x7557e207_64d0_4f84_9670_08b74f9d259d) 12, 26
2024-09-03 14:49:31.798 +02:00 INFO (de.stschnell/testMixTypeScriptExample) (temp/x7557e207_64d0_4f84_9670_08b74f9d259d)
    This is a multiline string.
    In TypeScript, we use backticks.
    It makes the code more readable.

2024-09-03 14:49:31.799 +02:00 INFO (de.stschnell/testMixTypeScriptExample) (temp/x7557e207_64d0_4f84_9670_08b74f9d259d) Current Direction is: 0
2024-09-03 14:49:31.800 +02:00 INFO (de.stschnell/testMixTypeScriptExample) (temp/x7557e207_64d0_4f84_9670_08b74f9d259d) Stefan
2024-09-03 14:49:31.801 +02:00 INFO (de.stschnell/testMixTypeScriptExample) (temp/x7557e207_64d0_4f84_9670_08b74f9d259d) https://[REDACTED].vmware.mgmt.rzsrv.svc:443/sdk

```

## Implementation (8)



# Updates

- Five release updates have been made since the initial implementation, from TypeScript 5.3.3 to 5.4.2, to 5.4.4, to 5.4.5, to 5.5.2 and to 5.5.4.
- The new TypeScript releases were tested with test cases in a simulation environment.
- The modules were then replaced and implemented.
- The new TypeScript release was then checked with test cases in Aria Automation.
- The expected time required is approximately from two to four hours.

# Runtime Environment Updates

- The TypeScript transpilation runs in the context of the Node.js execution environment.
- The Node.js execution environment has been upgraded to version 20 with the Aria Automation release 8.16.2.
- The new Node.js release was tested with the TypeScript release with test cases in a simulation environment.
- The settings of the actions of the Node.js execution environment were then adjusted.
- The new settings of the Node.js execution environment were then checked with test cases in Aria Automation.
- The expected time required is approximately from two to four hours.



# Findings (1)

- In summary, it can be said that the use of TypeScript in Aria Automation is seamlessly technically possible.
- Working with the Embedded Orchestrator makes handling in the development process more complicated.  
An action can no longer be started via Run, but via an additional action.
- Without JSDoc, it is clear from the source code which data types the variables uses.
- The use of const and let instead of var makes it clear how the variables are used in the code.
- The source code becomes clearer with TypeScript, because of the more modern language constructs that can be used.

# Findings (2)

- A real advantage can be seen with extensive complex programs that are not documented with JSDoc.
- The proportion of extensive, complex programs in Aria Automation is rather low. The structural concept of Aria Automation promotes the provision of atomic modules in the form of actions and scriptable tasks in workflows. This means that basic approaches are opposed.

# Findings (3)

- An analysis of the 677 JavaScript actions of the namespace com.vmware showed the following distribution in release 8.18:
  - 595 Actions have less than or 25 lines of code, 88%
  - 74 Actions have less than or 100 lines of code, 11%
  - 8 Actions have more than 100 lines of code, 1%Maximum 537 lines of code, counted without comments.
- An analysis of the 243 JavaScript actions of the namespace BWI showed the following distribution:
  - 89 Actions have less than or 25 lines of code, 37%
  - 112 Actions have less than or 100 lines of code, 46%
  - 42 Actions have more than 100 lines of code, 17%Maximum 329 lines of code, counted without comments.

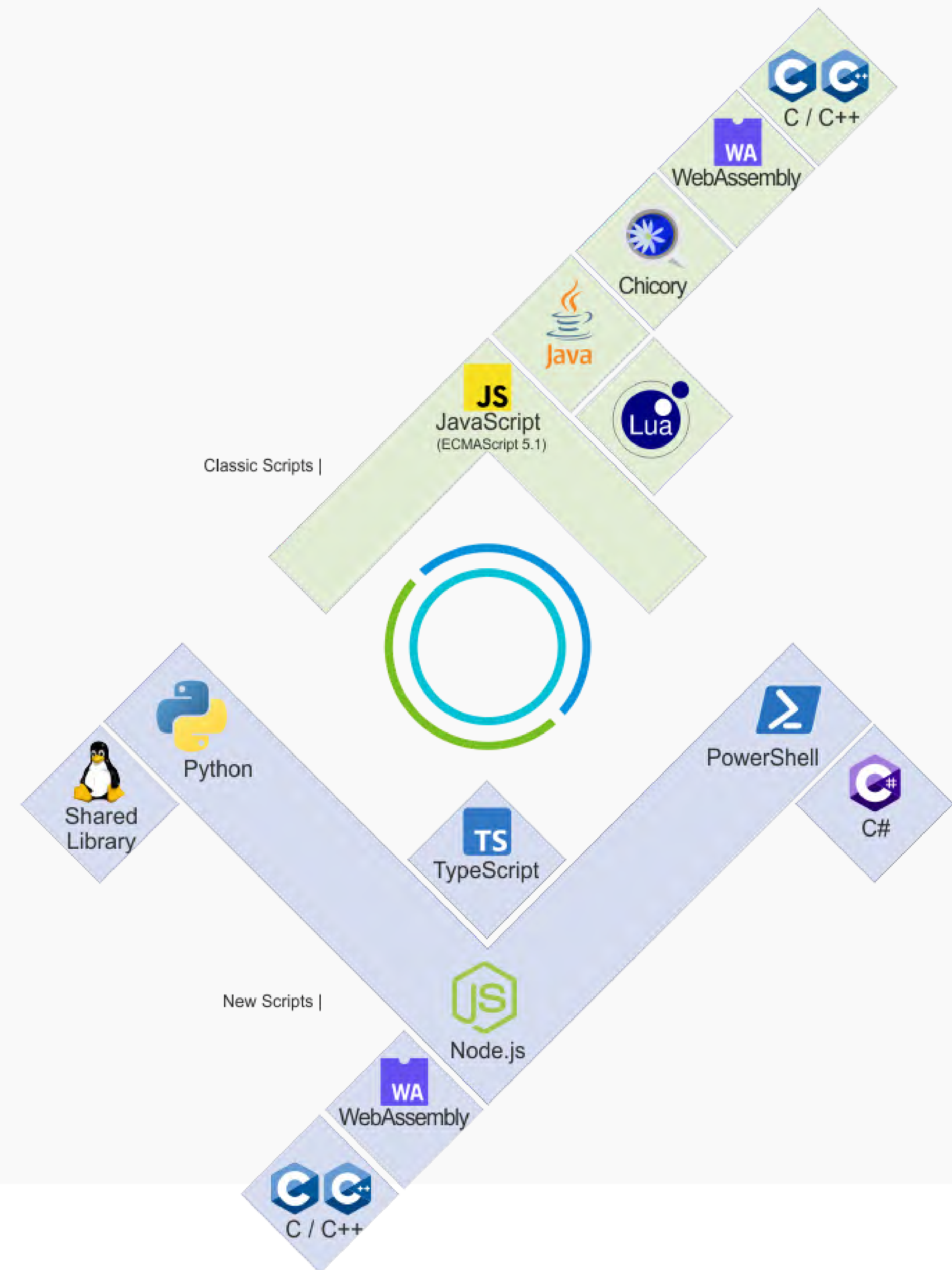


# Findings (4)

- An interface must be declared for each standard Aria Automation object, so that it can be used to transpile the TypeScript code into JavaScript code without errors. The initial effort required to build these interfaces is high.
- The return of a JSON that is embedded in an action log requires a considerably more complex conversion, to get the content for further processing.
- The use of a TypeScript action as an external value source of an element of an input form of a workflow is only possible to limits, because the return type of this TypeScript action is not compatible. But this can be achieved by using a wrapper.

# Findings (5)

- VMware Aria Automation can be enriched with many approaches, e.g. WebAssembly, here with C and C++, also possible with Rust, C#, Shared Object or Shared Library ...
- All the approaches shown here were realized in Embedded Orchestrator.



# Findings (6)

- The use of TypeScript will be discontinued, because ...
  - the complexity of the development process increases.
  - the advantages would only apply to a small proportion of JavaScript source codes in Aria Automation.



# Miscellaneous and Questions

- The next generation of a runtime for JavaScript, TypeScript and WebAssembly is focused, [Deno](#).  
A suggestion was submitted at 2024/04/10 in the VCF Feature Request Portal.  
It has the status of future consideration.
- All source codes presented here are available on GitHub:  
<https://github.com/StSchnell/Data-Center-Automation/tree/main/JavaScriptTools/TypeScript/addNumbers>
- Are there any comments or questions?



**Thank you very much  
for your attention**

**Stefan Schnell**  
BWI GmbH