ClusterSeven

ClusterSeven IMS 4.1

Guide to JavaScript rule code

ClusterSeven

Table of Contents

Legal Notice

Document Version History

Date	Changes made	Author
26th November 2019	Document created	Mayur Bapodra

Changes in 4.1

Changes were made to the validation of rule code in 4.1, which not only improves the security of the product as a whole, but also improves the quality of code that is used in a BU. This, in turn, makes it easier for PSO and support agents to read, remedy and maintain this code.

The two main changes made were:

- All code now runs in strict mode.
- JavaScript expressions should not end in a semi-colon, as this is syntactically incorrect.

We will set out the consequences of these changes below.

Consequences of changes

JavaScript code in rules is never updated as part of upgrades or updates – parsing, understanding and modifying rules in an automated way without introducing unwanted errors and bugs would be an almost intractable task.

As such, clients will need to update their rule code to make it compliant with the new requirements in 4.1. This document is intended as a guide to potentially necessary changes as well as existing common pitfalls. However, if further assistance is needed, please do not hesitate to refer to a member of the development team.

Strict mode

> All variables must be explicitly declared with “var”.

Previously, the following would have been valid:

```
function testFunction() {
	someVariable = 10;
}
```

This is no longer valid, and you will see an error that someVariable is not defined. The variable must be declared properly as follows:

```
function testFunction() {
	var someVariable = 10;
}
```

Variables have a well-defined scope.

Variables are scoped locally to the function in which they are defined. They are accessible to all child scopes. For example, in the following code segment, the parentVariable is defined in a function called parentFunction. This variable is only available within this function, and in any functions defined within this one (e.g., childFunction).

If variables are required in multiple places/functions, they can be defined “globally” by putting them outside of any function in the rule setup section (e.g., globalVariable).

```
var globalVariable = 8;
console.log(globalVariable); // OK, 8
console.log(parentVariable);  // error, parentVariable not available here

function parentFunction() {
	var parentVariable = 12;
	console.log(parentVariable);  // OK, 12

	function childFunction() {
		var childVariable = 14;
		console.log(parentVariable); OK, 12
		console.log(childVariable); OK, 14
	}
}

function otherFunction() {
	console.log(globalVariable); // OK, 8
}
```

Calculation Rules

In IMS, we have 4 kinds of rules – validation, editability, visibility and calculation. Calculation rules are the most complicated. The following general principles have not changed in 4.1, but are a guide to alleviate confusion about their use.

Calculation rule code expects a statement.

Calculation rules assign values to attribute/property targets using the keyword `value`. This is in the form of a **statement**. For example:

```
value = “hello”;
```

Statements represent an action or command. In the example above, this action is an assignment to the `value` variable. The “return” line in a function is also a statement, as it tells the function it is in to return a value.

Calculation rule code must assign to “value”.

As in the example above, to set the value of the rule target of the calculation rule, it must assign to the `value` keyword. The assignment can be a primitive data type or a variable:

```
value = “High Risk”;
value = parameters[“Attribute:Lightweight Process Model.Record.OpsRiskRating”];
```

It can also assign the result of a function evaluation that returns something. If we have the following function in the rule setup:

```
function getRiskRating() {
	var riskRating = “”;
	// do some calculations to work out the risk rating
	return riskRating;
}
```

then the following is also a valid value assignment:

```
value = getRiskRating();
```

Note that this can be written without the use of the function as:

```
var riskRating = “”;
// do some calculations to work out the risk rating
value = riskRating;
```

Calculation rule code statements *can* include a semi-colon, but it is not mandatory.

In all of the examples above, you may have noted that every statement ends in a semi-colon. This is not mandatory, but is advised to avoid inadvertent bugs. When semi-colons are not used, the JavaScript engine automatically tries to discern the ends of lines when running them. For example, the following will safely be run as expected without semi-colons:

```
var greeting = “hello”
return greeting
```

The engine is aware that the use of return starts a statement. The following, however, is more ambiguous:

```
var x = 1 + 1
-1 + 2
console.log(x); // prints 3, not 2
```

Without semi-colons, the first two lines are evaluated as a single statement (`var x = 1 + 1 - 1 + 2`), changing the intended value of x.

As a rule, it is better to terminate statements with a semi-colon to be on the safe side.

Non-calculation Rules

Validation, visibility and editability rules all require a yes/no evaluation to turn on/off functionality in a record. As such, they all expect their code to evaluate simply to true or false. Therefore, the code takes the form of an **expression**. An expression is a JavaScript snippet that evaluates to a value (in this case, a Boolean).

> Non-calculation rule code does not make use of the “value” keyword.

Non-calculation rules, therefore, do not use the `value` keyword we see in calculation rules, as no assignment is necessary.

> Non-calculation rule code does **NOT** terminate in a semi-colon.

The following is an example of an expression that might be used in an editability rule, setting the rule target as read-only if the risk rating of the record is high:

```
parameters[“Attribute:Lightweight Process Model.Record.RiskRating”] !== ‘High’
```

Semi-colons end statements. Remember that statements are actions or commands. Since expressions are essentially just values and not actions, it is fundamentally wrong to end an expression in a semi-colon. From IMS 4.1 onwards, non-calculation rules will be declared invalid if they break this rule.

> Non-calculation rule code can be any expression that evaluates to true or false.

Non-calculation rule code needs to boil down to either true or false once evaluated. As such it can also be the result of calling a function that returns true or false, or even an Immediately Invoked Function Expression (IIFE) that does so. An IIFE defines a function in place and then immediately invokes it. If the function returns a value, the entire IIFE simply equates to that value, hence the “expression” in its name.

The following are equally valid forms of the previous example, resulting in the same outcome:

In rule setup:

```
function isHighRiskRating() {
	return parameters[“Attribute:Lightweight Process Model.Record.RiskRating”] === 
‘High’;
}
```

Then, the non-calculation rule code:

```
!isHighRiskRating()
```

Using an IIFE, we could also write this without the use of the rule setup as:

```
!(function() {
	return parameters[“Attribute:Lightweight Process Model.Record.RiskRating”] === 
‘High’;
})()
```

Note that there is no semi-colon terminating the expression in both cases.

Contact

Europe

1st Floor, 27-32 Old Jewry
London, EC2R 8DQ, UK
T +44 207 148 6270
F +44 207 377 9124

North America

140 Broadway, 46th Floor
New York, NY 10005, USA
T +1 212 858 7790
F +1 212 858 7750

Website

www.clusterseven.com

Helpdesk

support.clusterseven.com