# **Multi-line rule executor**

Last Updated: 2024-02-20

A need often exists to build multi-line rules to compute a wanted value for an attribute. IBM® Security Verify offers the ability to configure both simple single-line and more advanced multi-line rules.

For single-line syntax rules, see [Attribute functions](https://www.ibm.com/docs/en/SSCT62/com.ibm.iamservice.doc/references/r_attr_functions.html).

## **Elements of a multi-line rule**

The multi-line rule is written in YAML format. It is constrained by the formatting needs of a YAML document. The YAML parser that is used supports most of YAML 1.1 and 1.2.

It is important to understand the following terms.

**Statement**

A statement is the building block of the rule and can be of different types. For example, the return statement computes the expression that was provided and returns the expression as the rule value.

**Block**

A block is a bounded collection of statements. The rule starts with a top-level block called statements. Blocks can exist within certain statement types. For example, the if.block is the collection of statements to be executed when the corresponding if.match statement evaluates to true.

**Block context**

A block context is a collection of variables that are available within that block. Subblocks can access the context of the parent blocks. Parent blocks cannot access the context of the subblocks.

**Expression**

An expression is a snippet that uses the same syntax as single-lined expressions. See [Attribute functions](https://www.ibm.com/docs/en/SSCT62/com.ibm.iamservice.doc/references/r_attr_functions.html).

## **Supported statement types**

Three types of statements are supported.

* Context
* Return
* If

## **Context statement**

A context statement is used to initialize and assign values to named variables. It supports two operators.

:=

This operator is used to initialize a new variable within the context block where the statement is evaluated. For example, if a variable is initialized within an if.block, it is not available outside this block in the subsequent statements.

=

This operator is used to assign a value to an existing variable. The variable can exist in the current context block or a parent block.

The format is context: {{varname}} := {{expression}}. Use is context.{{varname}} to access {{varname}} in subsequent statements in the same block or subblocks.

## **Return statement**

The return statement is used to indicate to the rule engine to end the execution and to immediately return the value that was computed.

The format is return: {{expression}}.

## **If statement**

The if statement is used to build conditional blocks. It consists of several properties, unlike the previous two statements.

| **Name** | **Description** | **Format** |
| --- | --- | --- |
| match | If this condition evaluates to true, the corresponding block is executed. | match: 1 == 2 |
| block | Bounded block of statements that must be executed if match evaluates to true. | A standard YAML array of statements |
| elsifs | Array of else-if conditions that are evaluated if match fails to evaluate to true. | YAML arrays that are nested if statements |
| else | Block of statements that are executed if match and all elseifs.match fail to evaluate to true. | A YAML array of statements |

## **Snippets**

This list of snippets is not exhaustive and does not cover all possible uses of rules. However, this list provides some samples that can be used as is or can be extended.

Note: Because the multi-line syntax follows the YAML format, correct indentation is crucial for the custom rule to function properly.

## **Get the manager's display name**

statements:

- context: manager := user.getManager()

- context: userExists := has(context.manager.name)

- context: >

givenNameExists := context.userExists

&& has(context.manager.name.givenName)

&& context.manager.name.givenName != ""

- context: familyNameExists := context.userExists && has(context.manager.name.familyName) && context.manager.name.familyName != ""

- context: formattedExists := context.userExists && has(context.manager.name.formatted) && context.manager.name.formatted != ""

- if:

match: context.formattedExists

block:

- return: context.manager.name.formatted

elseifs:

- match: context.givenNameExists

block:

- if:

match: context.familyNameExists

block:

- context: managerName := context.manager.name.familyName + ", " + context.manager.name.givenName

- return: context.managerName

else:

- return: context.manager.name.givenName

- match: context.familyNameExists

block:

- return: context.manager.name.familyName

- return: string("Not Available")

## **Transform the email domain**

statements:

- context: "workEmails := has(user.emails) ? user.emails.filter(e, e.type == 'work') : []"

- context: "workEmail := size(context.workEmails) > 0 ? context.workEmails[0].value : ''"

- if:

match: context.workEmail != ""

block:

- context: cn := context.workEmail.substring(0, context.workEmail.lastIndexOf('@'))

- if:

match: context.cn != ""

block:

- return: context.cn + "@github.com"

- return: ""

## **Scoping of variables**

statements:

- context: ret := 3 + 4

- if:

match: 2 > 1block:

- context: ret := 5

- if:

match: 3 > 2block:

- context: ret = 0

- return: context.ret

The result is 7. This result is because line 6 reinitializes ret within the if.block. The subsequent nested statements see this variable. However, after the rule engine exits the if.block, it can access the variable that was initialized at line 2.

## **Important restrictions**

* The variables that are initialized by the context frame can live only within that frame or subframes within it. In the previous rule example, the variable managerName is not accessible outside the if frame that it was initialized in.
* elseifs and else frames can be used only if an if frame is on the same level.
* All the if and elseifs frame must have a match expression that always returns a Boolean value.
* If the rule evaluation flow does not encounter a return statement, a null value is returned. No errors are thrown nor are syntax check performed to make sure that all possible flows have a return. The author of the rule must ensure that all flows of the rule return an acceptable value.