Luca Morgese Zangrandi, TNO, 2 April 2024

Contents

[Summary 2](#_Toc162952244)

[1 End Steps Modifications 3](#_Toc162952245)

[Concerns 3](#_Toc162952246)

[Possible ways to address 3](#_Toc162952247)

[2 On Completion, On Success, On Failure inconsistencies 4](#_Toc162952248)

[Concerns 4](#_Toc162952249)

[Possible ways to address 4](#_Toc162952250)

[Only change optional and required properties 4](#_Toc162952251)

[Change specification properties related to exceptions 5](#_Toc162952252)

[3 Variables Scope 7](#_Toc162952253)

[Concern 7](#_Toc162952254)

[Possible ways to address 7](#_Toc162952255)

[4 Variables properties 8](#_Toc162952256)

[Concerns 8](#_Toc162952257)

[Possible ways to address 8](#_Toc162952258)

[5 Out args management 9](#_Toc162952259)

[Concerns 9](#_Toc162952260)

[Possible ways to address 9](#_Toc162952261)

[6 Discussion on IDs and Playbook Sharing 10](#_Toc162952262)

[Concerns 10](#_Toc162952263)

[Possible ways to address 11](#_Toc162952264)

# Summary

This document presents modification proposals for the CACAO specification, listed approximately in order of growing complexity:

1. Introduce distinction between branch end step and workflow end step, and related end step properties extension.
	1. Main concern: back-tracking branching steps in complex logic.
	2. Main suggestion: introduce branch and workflow end distinction.
2. Resolve inconsistencies in the use of on completion, on success, on failure, and workflow exception.
	1. Main concern: all fields are optional and there is no exhaustive indication on how to use them, which may lead to inconsistent playbook execution.
	2. Main suggestion: remove unnecessary exception handling properties while defining a mandatory exception handling method.
3. Improve the scoping of variables in a workflow.
	1. Main concern: the only explicit scopes in the spec are playbook-level and step-level. Step-level scope may not be useful.
	2. Main suggestion: better define playbook-, step-, and branch-level scope.
4. Improve the variables definition and/or properties for a more effective variables substitution process.
	1. Main problem: many fields could be variables, while this does not necessary make sense.
	2. Main suggestion: introduce rules on variables and additional properties in variables objects
5. Define how out\_args should be handled.
	1. Main concern: the use of out\_args and how variables should be employed/populated in commands is unspecified. This may lead to inconsistent creation and execution of playbooks.
	2. Main suggestion: introduce command-specific out\_args and explicitly define expected out\_args assignment mechanisms.
6. Further define the mechanisms which support sharing of playbooks.
	1. Main concern: mandatory and expected ID properties may not be optimal to define playbooks and to share them across organizations and domains.
	2. Main suggestion: introduce valid “placeholder” ID values, specifically for playbooks (in playbook action steps), agent-targets, and authentication information.

For each modification, we first present in bullet points the elements that brought us to the modification proposal. Then, we list some thoughts on how the modification could be implemented.

# 1 End Steps Modifications

## Concerns

* Branching steps (if, while, parallel) have multiple branches: on\_true, on\_false, on\_completion, next\_steps.
* All branches must end in an end\_step.
* All branches end steps - except for on\_completion - should be taken as “end of the workflow branch, continue to the next workflow command” (details of on\_true in e.g. 4.8: “*This branch****MUST****reference a unique end step when that branch has completed processing. This allows implementations to know when to return to the original if condition step that started that branch to look for any****on\_completion****,****on\_success****, or****on\_failure****actions.*”)
* Workflow branch end steps should hence be used to back-track to the branching step (if, while, parallel), to then trigger the on\_completion (/success/failure) step.
* This requires that if a playbook has complex and nested conditional logic, all the conditional steps must be back-tracked by the implementation that executes the playbook. Hence, this introduces a great overhead for execution implementations.
* As related consideration, it may be the case that *the entire* *playbook execution* should terminate upon reaching a condition “nested deep” in the conditional logic, without having to backtrack the previous condition nodes.
* This is alike to using a return value inside nested or otherwise complex conditional logic in programming.

## Possible ways to address

* Introduce the concepts of playbook-end step and branch-end step.
* A “branch-end” step should provides necessary information for back-tracing.
* If a playbook reaches a playbook-end step, no more processing is needed. If a playbook reaches a branch-end step, the execution can be back-traced.
* Nice to note: this would be retro-compatible.

Example of a possible definition

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Data Type** | **Details** |
| **type** (required) | string | The value of this property **MUST** be end. |
| **branch\_origin** (optional) | identifier | This property **MAY** be used in end steps that terminate a workflow branch originated by **if-condition**, **parallel**, **while-condition**, or **switch-condition** steps. If specified, this property **MUST** point to the respective step ID of the step that originated the current branch. If specified, the property **playbook\_end** **MUST** be either set to false or empty. |
| **playbook\_end** (optional) | boolean | This property specifies if the reached end-step implies that the workflow execution should stop. If specified, the property **branch\_origin MUST** be empty. |

# 2 On Completion, On Success, On Failure inconsistencies

## Concerns

* According to the specification, a step handles exceptions in the following way
	+ Use on\_completion only
	+ Use on\_completion and define workflow\_exception
	+ Use on\_success together with on\_failure
	+ Use on\_success only (without specifying on\_failure),
	+ Use on\_success and define workflow\_exception
* There is no explicit indication on how to behave if an “on\_completion only” or “on\_success only” step “fails” (without there being a “workflow\_exception” defined).
* There is no explicit indication on how to behave if an on\_completion + workflow\_exception step “fails”, but this can happen:
	+ Throw the workflow\_exception
		- but this makes on\_completion superfluous: just use on\_success without on\_failure
	+ continue to the next step anyway
		- plausible, but not explicitly mentioned
	+ Either of the two, or up to the implementation
		- flexible, but it may create issues in executing a playbook inconsistently with two different CACAO execution implementations
* The specification says “The determination of a step being successful, failing, or completing is implementation specific and is out of scope for this specification”, but this also favours inconsistent behaviour of how to execute a CACAP playbook in two different playbook execution implementations, or human interpretation.

## Possible ways to address

### Only change optional and required properties

* Make mandatory use of workflow\_exception
* Make mandatory use of on\_failure if on\_success is used (and vice versa). In 4.1:

|  |  |  |
| --- | --- | --- |
| **on\_success** (optional) | identifier | The ID of the next step to be processed if this step completes successfully. The value of this property **MUST** represent a CACAO workflow step object. If this property is defined, then **on\_failure** **MUST** be defined, and **on\_completion** **MUST** **NOT** be defined. This property **MUST NOT** be used on the start or end steps. |
| **on\_failure** (optional) | identifier | The ID of the next step to be processed if this step fails to complete successfully. The value of this property **MUST** represent a CACAO workflow step object. DELETE: If omitted and a failure occurs, then the playbook’s exception handler found in the **workflow\_exception**property at the Playbook level will be invoked. If this property is defined, then **on\_success MUST** be defined, and **on\_completion** **MUST** **NOT** be defined. This property **MUST NOT** be used on the start or end steps. |

* Possibly, explicitly define what determines a step being successful, failing, or completing.
	+ Define what should represent a failure at least for all command types, e.g.:
		- an HTTP-API command returning non 2XX code SHOULD be deemed as step failure.
		- an SSH command returning a non-zero exit code SHOULD be deemed as step failure.
	+ Define how on\_completion should work, e.g.:
		- “the step pointed by on\_completion is executed next if the execution of the current step did not raise any exception”
	+ Define what is meant by step execution exception, e.g.:
		- “the execution of a(n action) step should raise an exception in any case that a system responds in a non-interpretable way”

### Change specification properties related to exceptions

Make mandatory use of workflow\_exception

* do not include the notion of “step failure” in the CACAO specification, and just use the concept of “workflow execution failure”, as a higher-level exception.
* In this direction, it could be considered to just remove on\_success, and substitute on\_failure with “on\_exception”, which overwrites the playbook\_exception, for a finer workflow management.
	+ A playbook would thus have workflow\_exception (required)
	+ A step would thus have on\_completion (required), and
	+ on\_exception (optional)

In 3.1 Playbook Properties

|  |  |  |
| --- | --- | --- |
| **workflow\_exception** (required) | identifier | The workflow step invoked whenever a playbook execution exception condition occurs.If defined, the ID **MUST** represent a CACAO workflow step object and that object **MUST** be included in the **workflow** property. |

4.1 Workflow step common properties

|  |  |  |
| --- | --- | --- |
| **on\_completion** (required) | identifier | The ID of the next step to be processed upon completion of the defined commands.The value of this property **MUST** represent a CACAO workflow step object. DELETE: If this property is defined, then **on\_success** and **on\_failure** **MUST** **NOT** be defined. |
| **on\_success** (optional) | identifier | The ID of the next step to be processed if this step completes successfully. The value of this property **MUST** represent a CACAO workflow step object. If this property is defined, then **on\_completion** **MUST** **NOT** be defined. This property **MUST NOT** be used on the start or end steps. |
| **on\_exception** (optional) | identifier | This property allows to define step-specific execution exception handling that overwrites the playbook-level **workflow\_exception** property.The ID of the next step to be processed if the execution of this step fails to complete successfully.The value of this property **MUST** represent a CACAO workflow step object.If this property is defined, then the playbook-level **workflow\_exception** **MUST** **NOT** be invoked. This property **MUST NOT** be used on the start or end steps. |

# 3 Variables Scope

## Concern

Observing that:

* playbook\_variables (3.1) define variables in playbook scope. step\_variables (4.1) define variables in step scope. These are the only two *explicit* variables scope in a playbook: global, or step-specific (10.18.1).
* All playbook steps can define step\_variables specific to that step, and which can be accessed by objects “used or referenced by that object” (10.18.1).
* in\_args are used in playbook action steps, or action steps. They are a list of string, i.e., the keys of the variables to resolve, and refer to variables used within that step or playbook.
* out\_args are also used in playbook action steps, or action steps. They are also a list of strings, i.e., the keys of the variables that are expectedly modified by the execution of the step.
* \_\_RETURN\_CALLER\_\_ (3.3) is defined as a constant pointing to the step that started a current branch

Note that:

* A workflow execution tool will always resolve/substitute variables in a step upon parsing the field that contains variables, hence, in\_args is potentially useless
	+ 🡪 The resolution/substitution of a variable can still be expected to first check the local step scope (step\_variables), then the global playbook scope (playbook\_variables)
* It does not make sense that out\_args refer to step-local variables, as once the step is executed, the step-local variables will be out of scope of the playbook – unless an executer expects step return values which are not relevant to the workflow, but rather of interest to the general execution environment.
	+ It makes sense if out\_args refer to playbook scope variables, as updating them will contribute information to the workflow
* IF and WHILE condition branches may only need step\_variables if their condition is not dependent on the workflow. If the condition is dependent on the workflow, then the variables used in the condition need to be global, as the condition changes based on the execution of previous steps, or consequent branch steps. (Alternatively, an additional playbook branch variables scope may be added).
* Why is \_\_RETURN\_CALLER\_ a constant, when there are many different workflow branches? Doesn’t it need to be updated dynamically?

## Possible ways to address

* Remove in\_args property
* Only make use of global\_scope variables.
* Alternatively, introduce explicitly a branch scope for variables for IF, WHILE and PARALLEL steps – though it may still be easier to only have playbook-level variables.
* Make \_\_RETURN\_CALLER\_\_ a mutable variable

# 4 Variables properties

## Concerns

* It does not make sense for all properties in a playbook to be potential variables, or to “accept” variables use (e.g. names, descriptions, civic locations, …).
* This implies that implementations should check all property field values if either they are variables, or include variables.
* This causes some implementation complexity that should be avoidable.

## Possible ways to address

* Define/indicate what fields may not be or host variables (e.g., “name” property should always be defined).

Possibly, add optional properties for a variable object to advertise in which part or objects of the playbook the variable is used.

# 5 Out args management

## Concerns

* Variable referencing in Example 4.8 (pasted below) is possibly deceiving due to undefined management of variables: \_\_userIn\_\_ is used as “expected” returned variable from an ssh command, within the ssh command, but there is no suggestion or notion of how this would work. This may be confusing to implement and lead to diverging playbook commands and variables management implementation strategies, which would not be interoperable.

    "action--fc7bda8b-c2d8-4533-b022-19a68e150e44" : {

      "type": "action",

      "name": "Ask the user to choose the next step via cli command"

      "commands": [

        {

          "type": "bash",

          "command": "/opt/bin/ask\_cli\_question(\"Enter next step choice (1, 2, other) ?\", \"\_\_userseln\_\_\")",

        }

      ],

      "on\_completion": "switch-condition--33ba061e-193d-41db-b40b-0e8373997dc9",

      "out\_args": [ "\_\_userseln\_\_" ]

* What if one step has more out\_args than commands? Or what if it has less out\_args than commands? How are args/variables managed and assigned?
* An unclear indication on how and where to write and resolve variables and out\_args may lead to inconsistent playbook definitions and executions for different implementations.

## Possible ways to address

* Remove \"\_\_userseln\_\_\" in the command from example 4.8
* A possible way to address the condition of a step having more out\_args than commands, and vice versa, may be distinguishing step\_out\_args, and command\_out\_arg (note singular).
* With such distinction, command\_out\_arg contain only one variable which must be populated entirely with whatever information is returned by the execution of the command.

# 6 Discussion on IDs and Playbook Sharing

## Concerns

Issue with sharing:

* At present, all IDs need to be defined for a playbook to be valid. With the intention of sharing a playbook, this may be a problem, as a playbook-producing company may not want/need to share IDs of their CACAO objects, and a playbook-consuming company may want to use their own agent/target/playbooks in action and playbook steps, referenced with own IDs.
* A company, or a national CERT, may want to structure playbooks in a hierarchical way, and re-use low- and high-level playbooks to compose new or updated playbooks. For instance, there might be a very generic playbook step which indicates to“block an offending domain”**.** This action may be performed in very different ways depending on what security devices can be used to perform the action (firewalls, routers, …), where in the infrastructure such devices are, what C2 protocols can be used to instruct the devices, and how much automation an organization is capable/willing to use.

Observations and discussion towards effective sharing of CACAO playbooks:

* It should be possible to provide a CACAO playbook in a “generalized, parametric” form which fulfils schema validation, but allows specific contexts to populate the playbook such that it can be executed.
* For the sake of argument, consider a distinction between higher level “effect” playbooks (e.g.: block host’s internet access), and lower level “execution” playbooks (e.g.: update CISCO firewall rule via SDN controller on a specific subnet).
* Effect playbooks should be generalized, while the achievement (execution) of a given effect may be performed in specific ways according to capabilities, technologies, context.
* This would favor effect playbooks being organised in hierarchies, abstractions, taxonomies.
* Effect playbooks steps should support plugging any execution steps. This means that the step of an effect playbook should support diverse execution modalities – i.e. different action types.
	+ 🡪 This is probably best achieved via making every effect playbook step of type playbook execution step, where organizations could plug in specific playbooks that realize an effect.
* Both effect and execution playbooks should be shareable.
	+ In some cases, effect playbooks may present a general step-by-step procedure to e.g. bring an IT infrastructure to a given operational status. In these cases, companies should only receive effect playbooks, and realize them via instantiating/choosing and deploying specific execution playbooks that implement the higher-level action how best they can with their capabilities.
	+ In other cases, effect playbooks may be already accompanied by execution playbooks that achieve the given effects, for instance in case of patching a specific bug of a specific firmware (once the device is located).
* Effect playbooks steps should not need to be parametrized, especially with the consideration that they only keep playbook execution steps.
* The steps in an execution playbook need to be parametrized with respect to: agents and their properties, targets and their properties, variables in commands, authentication information.
* Execution playbooks do not need to be generalized for different technologies. Each execution playbook is specific to a technology, but parametric in the technology context properties (location, access, identifiers, …).

## Possible ways to address

* Define a well-known “placeholder identifier” that is correctly validated.
	+ A rough idea could be allowing variables for properties which should be identifiers. See next point.
* The placeholder identifier may be used for:
	+ the playbook\_id identifier in the playbook action step, intended for effect playbooks
	+ agent-targets ids in action steps,
	+ authentication information for agent-targets (for when agent-targets are defined)
* Placeholder/Stub IDs: would allow for validation of non-populated shareable or generalized playbooks, hence allowing a “dynamic instantiation” of referenced playbooks for playbook action steps, agent-targets and related properties