Configure YAML Overrides in Config Sets

Last Modified on 08/19/2025 8:00 am EDT

V10.8

Overview

As a workflow developer, while creating a new config set, you can set an existing config set as a base to inherit it's configuration and override necessary configuration with minimal changes. The functionality improves the implementation practices and facilitates upgrade procedures.

Overriding YAML documents

Following is the common structure of YAML documents in a config set in Cora Orchestration:

```
kind: <document|ruleSet>
metadata:
  name: <document name>
  spec: <document content>
```

At runtime, the documents are identified by the `name` parameter under the `metadata`. When a config set is linked to the base config set, the documents in the inherited config set with same `name`, will replace the documents from the base config set.
For example, if base config set contains a document with `name: config-test/v1/documents/test`, and the user adds a document in the inherited config set with the same `name`, the document from the base config set will be ignored.

Overriding YAML document contents

If the base config set contains the YAML document that is defined in the common structure, you can merge its contents with a YAML document from the inherited config set.
Add `/$overrides` at the end of the `name` property of the document.

For example, given the following document defined in the base config set:

```
kind: document
metadata:
  name: config-test/v1/documents/test
spec:
  - item1: value1
  - item2: value2
```

The user may define a document in the inherited config set with `/$overrides` at the end of the document `name`.

```
kind: document
metadata:
  name: config-test/v1/documents/test/$overrides
spec:
  - item3: value3
```

As a result, the two documents will be merged:

```
kind: document
metadata:
  name: config-test/v1/documents/test
spec:
  - item1: value1
  - item2: value2
  - item3: value3
```

YAML merge logic

When two YAML documents are merged, several rules apply recursively to all nodes of the document (starting on the `spec` node). The merge logic depends on the node type.

Mapping Nodes

For a mapping node (key-value set), the values from the base document will be replaced with the values from the override document.

Document in the base config set:

```
kind: document
metadata:
  name: config-test/v1/documents/test
spec:
  prop1: value1
  prop2: value2
```

Document in the inherited config set:

```
kind: document
metadata:
  name: config-test/v1/documents/test/$overrides
spec:
  prop2: newValue2
```

Result:

`prop1` keeps the original value, `prop2` takes the value from the override document:

```
kind: document
metadata:
  name: config-test/v1/documents/test
spec:
  prop1: value1
  prop2: newValue2
```

Sequence Nodes - Keyed Collections

For a sequence node (array) that contains a collection of mapping nodes, the system checks whether the child nodes contain a property named `id`, `name` or `$key`. If such property exists, the child node is considered a keyed item, and the merge will take the keys into account.

Document in the base config set:

Note that first item under `prop1` doesn't have the `name`, it will not be affected by the merge.

```
kind: document
metadata:
  name: config-test/v1/documents/test
spec:
  prop1:
    - value: sub1val
    - name: sub2
      value: sub2val
      subItems:
       - item1
       - item2
       - item3
  prop2: value2
```

Document in the inherited config set:

```
kind: document
metadata:
  name: config-test/v1/documents/test/$overrides
spec:
  prop1:
    -name: sub2
      value: newSub2val
      subItems:
        - item4
```

Result:

The item `sub2` under `prop1` is merged with the same item from the base document. Note that the override document updated the `value` property, and merged the collections under `subItems`:

```
kind: document
metadata:
  name: config-test/v1/documents/test
spec:
  prop1:
    -value: sub1val
    -name: sub2
      value: newSub2val
      subItems:
       - item1
       - item2
       - item3
       - item4
  prop2: value2
```

Using YAML tags for advanced merging

Clear

The tag `!clear` allows to clear the collection under the sequence node. This tag works both on keyed and non-keyed collections.

Document in the base config set:

```
kind: document
metadata:
  name: config-test/v1/documents/test
spec:
  prop1:
    -name: sub1
      value: sub1val
    -name: sub2
      value: newSub2val
      subItems:
       - item1
       - item2
       - item3
       - item4
  prop2: value2
```

Document in the inherited config set:

```
kind: document
metadata:
  name: config-test/v1/documents/test/$overrides
spec:
  prop1:
    - !clear
```

Result:

The `prop1` contains no items:

```
kind: document
metadata:
   name: config-test/v1/documents/test
spec:
   prop1: []
   prop2: value2
```

Insert After/Insert Before/Insert At

The tags `!insertAfter` and `!insertBefore` adds an item after or before another in the collection. The tag must be placed in a separate property named `$sequence`. This tag works only with keyed collections.

Similarly, the `!insertAt` tag adds an item to a specific position in the list.

Document in the base config set:

```
kind: document
metadata:
  name: config-test/v1/documents/test
spec:
  prop1:
    - name: first
      value: firstVal
    - name: last
      value: lastVal
  prop2: value2
```

Document in the inherited config set:

```
kind: document
metadata:
  name: config-test/v1/documents/test/$overrides
spec:
  prop1:
    - name: second
      value: secondVal
      $sequence: !insertAfter first
```

Result:

Under the `prop1`, the `second` is inserted after the `first`:

```
kind: document
metadata:
  name: config-test/v1/documents/test
spec:
  prop1:
    - name: first
      value: firstVal
    - name: second
      value: secondVal
    - name: last
      value: lastVal
  prop2: value2
```

Remove/Remove At

Tags `!remove` and `!removeAt` remove the keyed item from the collection. Use `!remove <name>` to delete item by name, or `!removeAt <position>` to remove item at a specific position.

Document in the base config set:

```
kind: document
metadata:
  name: config-test/v1/documents/test
spec:
  prop1:
    - name: first
      value: firstVal
    - name: last
      value : lastVal
  prop2: value2
```

Document in the inherited config set:

```
kind: document
metadata:
  name: config-test/v1/documents/test/$overrides
spec:
  prop1:
    - !remove first
```

Result:

The item named `first` is removed from `prop1` collection.

```
kind: document
metadata:
  name: config-test/v1/documents/test
spec:
  prop1:
    - name: last
      value: lastVal
  prop2: value2
```

NOTE

- Configure inheritance:
  - Setting a base config set is done via the *Edit Properties* dialog in the DocumentSet Editor.
  - Config set inheritance is supported in one level only. You may set a config set as base only if that config set is not already inheriting another config set.
- Case sensitivity:
  - The `name` under `metadata` tag is case-sensitive.
  - The value of `name` , `id` , or `$key` property in the nodes list is case-insensitive.
- Conflict resolution:
  - If you use `!remove` or `!removeAt` tag with wrong name and index, it will be ignored.
  - If there are a few items with the same `name` , `id` , or `$key` in the base, the override will affect only the first item. Second and rest items with the same name will always be added to the result of the override.
  - If you add a few items with the same `name` , `id` , or `$key` in the override (given there is an item with this name in the base), it will merge all those items sequentially with the item from the base.
  - The priority for key lookup is the following: first it looks for `$key` property, if it does not exist, it checks for `name` property, and lastly for the `id` property.