# Variables

Variables are part of a workflow instance and represent the data of the instance. A variable has a name and a JSON value. The visibility of a variable is defined by its variable scope.

## Variable Values

The value of a variable is stored as a JSON value. It must have one of the following types:

- String
- Number
- Boolean
- Array
- Document/Object
- Null

## Access Variables

Variables can be accessed within the workflow instance, for example, on input/output mappings or conditions. In the expression, the variable is accessed by its name. If the variable is a document then the nested properties can be accessed via dot notation.

Examples:

| Variables | Expression | Value |
|---|---|---|
| `totalPrice: 25.0` | `totalPrice` | `25.0` |
| `order: {"id": "order-123", "items": ["item-1", "item-2"]}` | `order` | `{"id": "order-123", "items": ["item-1", "item-2"]}` |
| `order: {"id": "order-123"}` | `order.id` | `"order-123"` |
| `order: {"items": ["item-1", "item-2"]}` | `order.items` | `["item-1", "item-2"]` |

## Variable Scopes

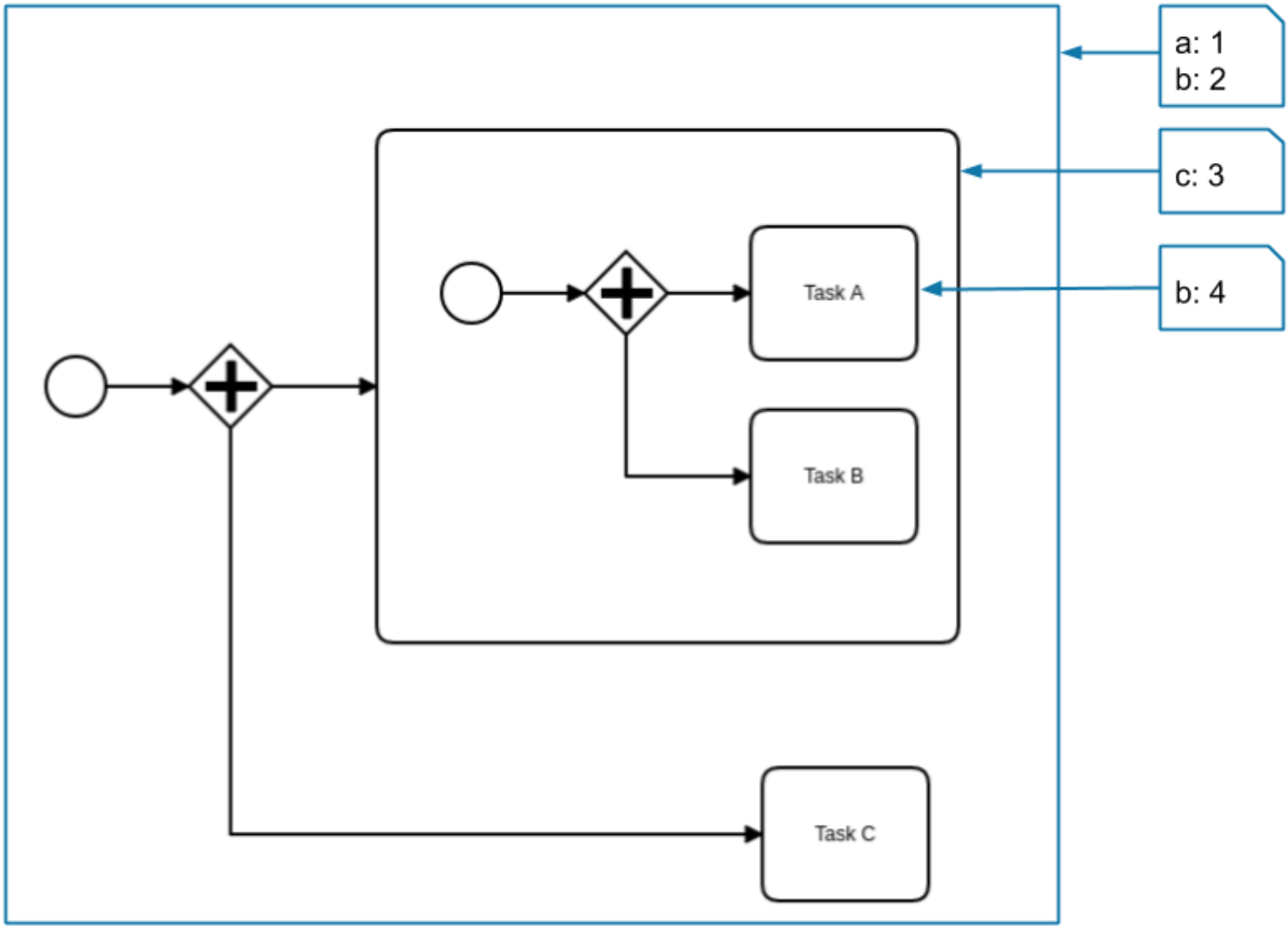
Variable scopes define the visibility of variables. The root scope is the workflow instance itself. Variables in this scope are visible everywhere in the workflow.

When the workflow instance enters a sub process or an activity then a new scope is created. Activities in this scope can see all variables of this and of higher scopes (i.e. parent scopes). But activities outside of this scope can not see the variables which are defined in this scopes.

If a variable has the same name as a variable from a higher scope then it covers this variable. Activities in this scope see only the value of this variable and not the one from the higher scope.

The scope of a variable is defined when the variable is created. By default, variables are created in the root scope.

Example:




This workflow instance has the following variables:

- `a` and `b` are defined on the root scope and can be seen by *Task A*, *Task B*, and *Task C*.
- `c` is defined in the sub process scope and can be seen by *Task A* and *Task B*.
- `b` is defined again on the activity scope of *Task A* and can be seen only by *Task A*. It covers the variable `b` from the root scope.

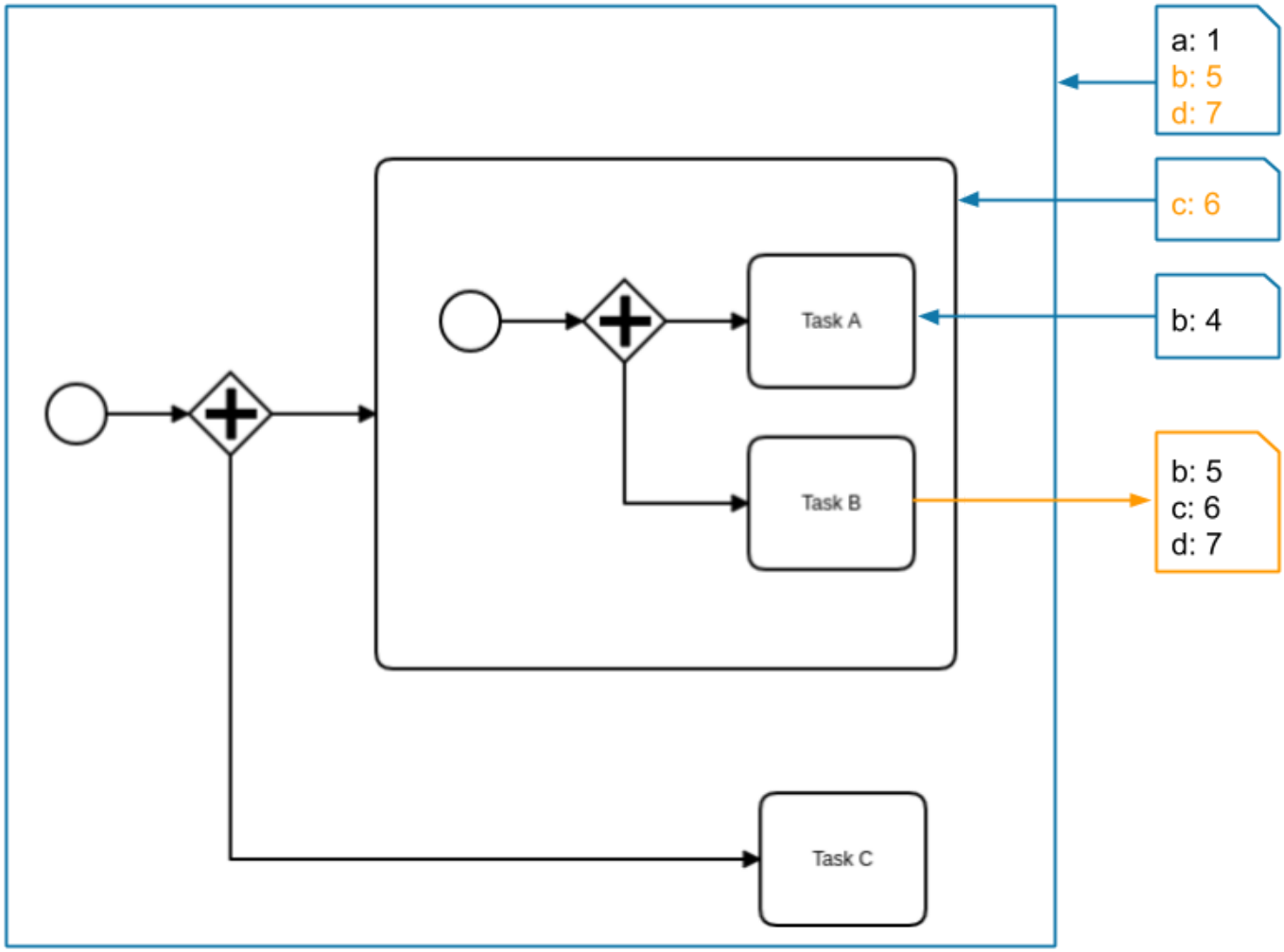
## Variable Propagation

When variables are merged into a workflow instance (e.g. on job completion, on message correlation) then each variable is propagated from the scope of the activity to its higher scopes.

The propagation ends when a scope contains a variable with the same name. In this case, the variable value is updated.

If no scope contains this variable then it is created as a new variable in the root scope.

Example:




The job of *Task B* is completed with the variables `b`, `c`, and `d`. The variables `b` and `c` are already defined in higher scopes and are updated with the new values. Variable `d` doesn't exist before and is created in the root scope.

## Local Variables

In some cases, variables should be set in a given scope, even if they don't exist in this scope before.

In order to deactivate the variable propagation, the variables are set as *local variables*. That means that the variables are created or updated in the given scope, whether they exist in this scope before or not.

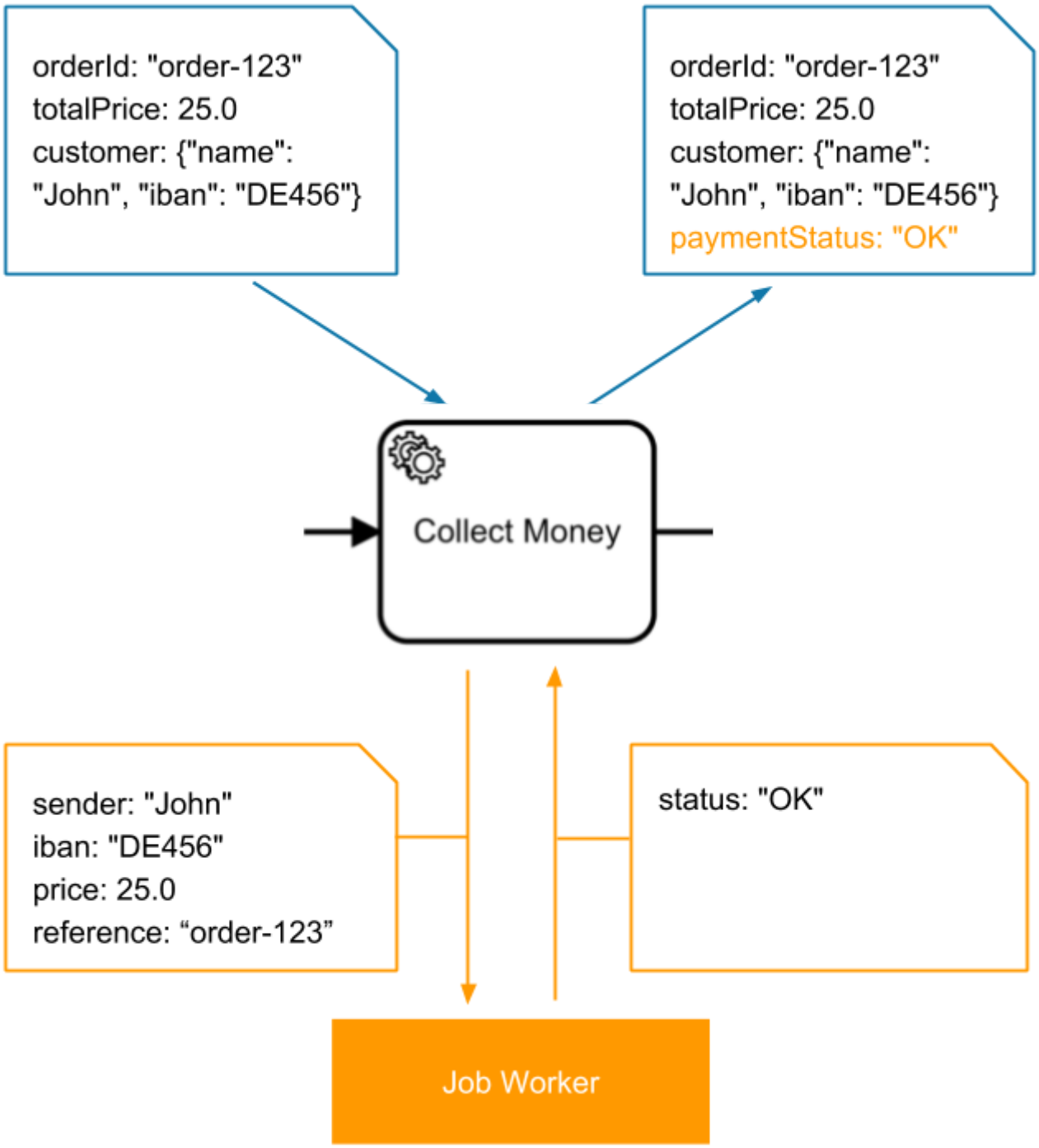
## Input/Output Variable Mappings

Input/output variable mappings can be used to create new variables or customize how variables are merged into the workflow instance.

Variable mappings are defined in the workflow as extension elements under `ioMapping`. Every variable mapping has a `source` and a `target` expression. The *source* expression defines where the value is copied from. The *target* expression defines where the value is copied to. The expressions reference a variable by its name or a nested property of a variable.

If a variable or a nested property of a *target* expression doesn't exist then it is created. But if a variable or a nested property of a *source* expression doesn't exist then an incident is created.

Example:




XML representation:

```
<serviceTask id="collectMoney" name="Collect Money">
    <extensionElements>
      <zeebe:ioMapping>
        <zeebe:input source="customer.name" target="sender"/>
        <zeebe:input source="customer.iban" target="iban"/>
        <zeebe:input source="totalPrice" target="price"/>
        <zeebe:input source="reference" target="orderId"/>
        <zeebe:output source="status" target="paymentStatus"/>
       </zeebe:ioMapping>
    </extensionElements>
</serviceTask>
```

## Input Mappings

Input mappings can used to create new variables. They can be defined on service tasks and sub processes.

When an input mapping is applied then it creates a new variable in the scope where the mapping is defined.

Examples:

| Workflow Instance Variables | Input Mappings | New Variables |
|---|---|---|
| `orderId: "order-123"` | `source: orderId`<br>`target: reference` | `reference: "order-123"` |
| `customer: {"name": "John"}` | `source: customer.name`<br>`target: sender` | `sender: "John"` |
| `customer: "John"`<br>`iban: "DE456"` | `source: customer`<br>`target: sender.name`<br><br>`source: iban`<br>`target: sender.iban` | `sender: {"name": "John",`<br>`"iban": "DE456"}` |

## Output Mappings

Output mappings can be used to customize how job/message variables are merged into the workflow instance. They can be defined on service tasks, receive tasks, message catch events and sub processes.

If one or more output mappings are defined then the job/message variables are set as *local variables* in the scope where the mapping is defined. Then, the output mappings are applied to the variables and create new variables in this scope. The new variables are merged into the parent scope. If there is no mapping for a job/message variable then the variable is not merged.

If no output mappings are defined then all job/message variables are merged into the workflow instance.

In case of a sub process, the behavior is different. There are no job/message variables to be merged. But output mappings can be used to propagate *local variables* of the sub process to higher scopes. By default, all *local variables* are removed when the scope is left.

Examples:

| Job/Message Variables | Output Mappings | Workflow Instance Variables |
|---|---|---|
| `status: "Ok"` | `source: status`<br>`target: paymentStatus` | `paymentStatus: "OK"` |
| `result: {"status": "Ok",`<br>`"transactionId": "t-789"}` | `source: result.status`<br>`target: paymentStatus` | `paymentStatus: "Ok"`<br>`transactionId: "t-789"` |

| | | |
|---|---|---|
| | source: result.transactionId target: transactionId | |
| status: "Ok" transactionId: "t-789" | source: transactionId target: order.transactionId | order: {"transactionId": "t-789"} |