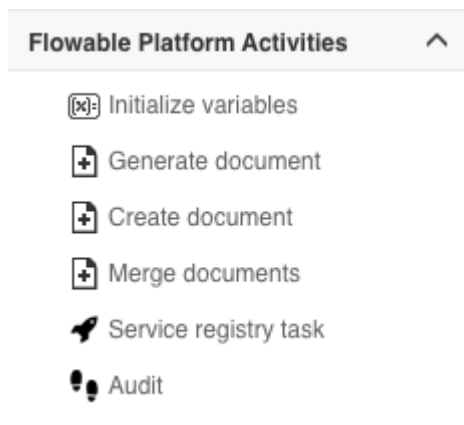


Initialize Variables Task

Introduction

The **Initialize Variables** task is an activity for Flowable Work that is available in the BPMN and CMMN palette in Flowable Design. It can be found under the *Flowable Work Activities* group in the palette:



The purpose of this task is clear from the name: it allows to initialize process or case variables with certain values when executed. The task can also be used to prepare the structure of more complex variables (like JSON variables).

When dropping the task on the canvas, two properties specific to this task type are shown in the right-hand property panel: *Init variables* and *Overwrite if existing*:

Clicking the *Init variables* property will open a popup window that allows to set and configure various variables:

A screenshot of the 'Init variables' popup window. The title bar says 'Init variables' with a close button. The window contains a table with four columns: 'Target', 'Name', 'Value Type', and 'Value'. The 'Target' column has a trash icon and up/down arrows. The 'Name' column has a text input field. The 'Value Type' column has a dropdown menu currently set to 'String'. The 'Value' column has a text input field and a lightning bolt icon. Below the table is a blue button labeled '+ Add Variable'. At the bottom right are 'Cancel' and 'OK' buttons.

The *Overwrite if existing* checkbox, when checked, determines whether or not to overwrite any existing variable with the variable values set in this task. The default is true: existing variables will be overwritten.

Configuration

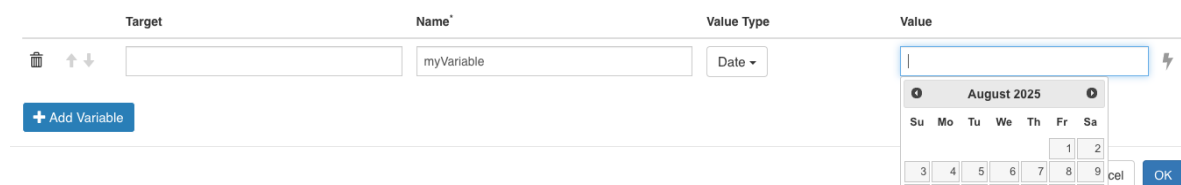
The *Init variables* popup has a tabular format: each row corresponds with a variable (or, as we'll see later, the definition of the structure of a complex variable).





The icons on the left hand-side can be used to respectively remove, move up or move down that row. Moving the rows up and down doesn't make a difference for regular variables, but the order is important for complex variables as the rows are read from top to bottom at runtime.


The **Target** column is optional. It can be used when defining a complex variable JSON structure (see later) or with values *root* or *parent*. Setting *root* or *parent* will set the variable not on the local instance, but on the parent or root instance in a nested hierarchy of process/case instances.

The **name** column is used to determine the name of the variable and is mandatory.

The **Value Type** column is used to determine the type of the variable, which is important as it switches the input field type depending on the chosen type (for example a date picker when selecting *Date*):



Target	Name	Value Type	Value
  	myVariable	Date	<input type="text" value=""/> 

 Add Variable

Current supported value types are:

- Primitive types that set a variable with the corresponding type: *Boolean*, *Double*, *Integer*, *Long* and *String*.
- *JSON Array/Object* for defining JSON variable values or structures. More about it later.
- *Variable*: Used to copy the value from another value.

The **Value** column allows to set the actual value and the input field type for filling in the value depends on the selected *Value type*, as shown above.

Next to the *Value* input field is a lightning icon. When clicked, the current row changes:

- The *Value* field becomes an input text area that allows to write an arbitrary expression.
- The *Value type* dropdown disappears, as the type will now implicitly be determined by the result of the expression.

This option is meant for advanced variable initializations and is used to write potentially complex expressions for use cases that cannot be covered by the default configuration options. As this is a back-end expression, all the usual beans are available.

Init variables ✕

Target	Name*	Value Type	Value
<div> <div>✕</div> <div>↑ ↓</div> </div> <input type="text"/>	myVariable	<div>▼</div>	<div> <div>⚡</div> <div> <code>\${myService.calculateVariable(existingVariable)}</code> </div> </div>

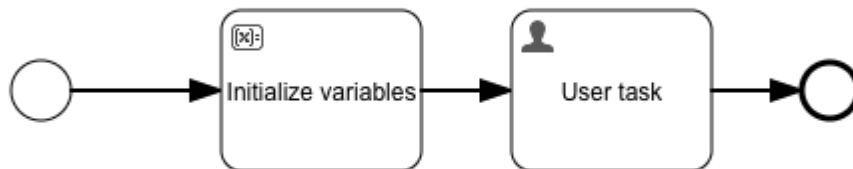
+ Add Variable

Cancel

OK

Basic Example



















Let's build a simple BPMN process to get familiar with the *Initialize variables* task. Drop both an *Initialize variables* and *User Task* to the canvas and connect them with sequence flow:



For the *Init variables* property, let's add some variables:

- firstName (String)
- lastName (String)
- married (Boolean)
- street (String)
- number (Integer)
- city (String)

Add some random values for each variable. You'll notice that it's for example not possible to write regular text in the *number* field, because the *value type* is set to *Integer*.

Target	Name*	Value Type	Value
  	<input type="text" value="firstName"/>	String ▾	<input type="text" value="John"/> ⚡
  	<input type="text" value="lastName"/>	String ▾	<input type="text" value="Doe"/> ⚡
  	<input type="text" value="married"/>	Boolean ▾	<input checked="" type="checkbox"/> ⚡
  	<input type="text" value="street"/>	String ▾	<input type="text" value="Randomstreet"/> ⚡
  	<input type="text" value="number"/>	Integer ▾	<input type="text" value="123"/> ⚡
  	<input type="text" value="city"/>	String ▾	<input type="text" value="Doeville"/> ⚡

[+ Add Variable](#)

[Cancel](#) [OK](#)

NOTE

The use case here is not necessarily to set values of variables hard-coded to a given value (that's not often useful). The main purpose is to 1) make sure the required variables exist with the proper type and 2) (optionally) that default values for them are provided.

For the user task, create a form that binds the values to the variables set in the *Init variables* (for example the field with label *First Name* is bound to value `{{firstName}}`):

First Name <input type="text" value="{{firstName}}"/>	Last Name <input type="text" value="{{lastName}}"/>	Married <input type="checkbox"/>
Street <input type="text" value="{{street}}"/>	Nr. <input type="text" value="{{number}}"/>	City <input type="text" value="{{city}}"/>

Publish this process model. When now starting a new process instance and opening the user task, we can see the values have indeed correctly been initialized:

Task

User task

initVariablesProcess

Created: less than a minute ago

Assignee: Flowable Admin

Due: No due date

Complete

Save

Task

Conversation

Comments

People

Subtasks

Documents

History

First Name <input type="text" value="John"/>	Last Name <input type="text" value="Doe"/>	<input checked="" type="checkbox"/> Married
Street <input type="text" value="Randomstreet"/>	Nr. <input type="text" value="123"/>	City <input type="text" value="Doeville"/>

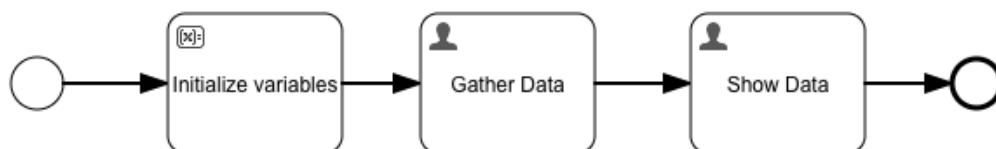
When running with Flowable Inspect enabled, you can see each row in the popup above corresponds with one variable created at runtime:

Test Users	Inspect	Current Test	Available Tests	Current form
User task TSK-d95c5317-4390-11eb-9ac1-acde48001122				
Execution tree Search initVariablesProcess User task				
Variables				
Evaluate expression now				
Scope	Name	Type	Value	
	firstName	string	John	
	number	integer	123	
	lastName	string	Doe	
	city	string	Doeville	
	street	string	Randomstreet	
	initiator	string	admin	
	married	boolean	true	

JSON Variable Example

Another use case for the *Initialize variables* task is to define the structure of more complex JSON variables.

To demonstrate this, let's tweak the model from the [previous section](#). The BPMN model now gets an additional task, which we'll call *Show Data*. We'll rename the original user task to *Gather Data*:



The main difference with the [basic example](#) is that we don't want to store one variable for each form field. Instead, we'd like to use the JSON variable type to stores all information in one variable. Change the *Init variables* property as follows:

- Add a new row with name *person* at the top with value type **JSON Object**. This will create a JSON variable with name *person*. The next rows will now be used to define the structure of the *person* variable.
- Add *person* to all other fields in the target column. As *person* is a JSON object, setting the *target* in this way instructs the engine to put those fields on the json object at runtime.
- Add a prefix *address.* to the street, number and city variables. This way, we automatically create a nested json object within the *person* variable.
- Remove all values.

After doing this, the configuration should look as follows:

	Target	Name	Value Type	Value
🗑️ ⬆️ ⬆️		person	JSON Object ▾	
🗑️ ⬆️ ⬆️	person	firstName	String ▾	
🗑️ ⬆️ ⬆️	person	lastName	String ▾	
🗑️ ⬆️ ⬆️	person	married	Boolean ▾	<input type="checkbox"/>
🗑️ ⬆️ ⬆️	person	address.street	String ▾	
🗑️ ⬆️ ⬆️	person	address.number	Integer ▾	
🗑️ ⬆️ ⬆️	person	address.city	String ▾	

+ Add Variable

Cancel OK

The goal is to have a JSON variable named `person` with following structure:

```
{
  "firstName": "a",
  "lastName": "b",
  "married": false,
  "address": {
    "street": "c",
    "city": "d",
    "number": 0
  }
}
```

NOTE

We're not setting any default values now (but we could, if we wanted to). The main purpose here is defining the structure of the JSON variable.

As we're using a JSON variable now, we need to bind form fields using `{{person.x}}` now in the form of the *Gather Data* task. For the street number and city, we need to use `{{person.address.x}}`:

First Name:

Last Name:

Married: ☐ {...}

Street:

Nr.: #

City:

Associate the same form with the *Show Data* task. In the first user task form we will fill in the fields and in the second task we'll view them.

Publish this new model and start a new instance. None of the form fields of the *Gather Data* task will have values set (as no value was set above):

Task

Gather Data

Created: less than a minute ago
Assignee: Flowable Admin
Due: No due date

Complete Save

Task Conversation Comments People Subtasks Documents History

First Name Last Name ☐ Married

Street Nr. City

Fill in some values for the various fields and click the *Complete* button. Select the new *Show Data* task.

You'll see that the values from the previous form are shown:

Task

Show Data

Created: less than a minute ago
Assignee: Flowable Admin
Due: No due date

Complete Save

Task Conversation Comments People Subtasks Documents History

First Name Last Name ☒ Married

Street Nr. City

Jane Doe Other Street 987 Doeville

When running with Flowable Inspect enabled, you can see the difference with the [basic example](#) clearly; instead of having multiple variables, theres only one *person* variable:

Test Users Inspect Current Test Available Tests Current form

Show Data
TSK-e32e6078-4389-11eb-9ac1-acde48001122

Variables Evaluate expression now

Scope	Name	Type	Value
	person	json	{"firstName":"Jane","last...
	initiator	string	admin

Execution tree

Search

initVariablesProcess
Show Data
Gather Data

Clicking the edit icon, shows that all data is now stored as a JSON variable with nested properties:

Edit 'person'

```
1 {  
2   "firstName": "Jane",  
3   "lastName": "Doe",  
4   "married": true,  
5   "address": {  
6     "street": "Other Street",  
7     "city": "Doeville",  
8     "number": 987  
9   }  
10 }
```

Ln: 1 Col: 1

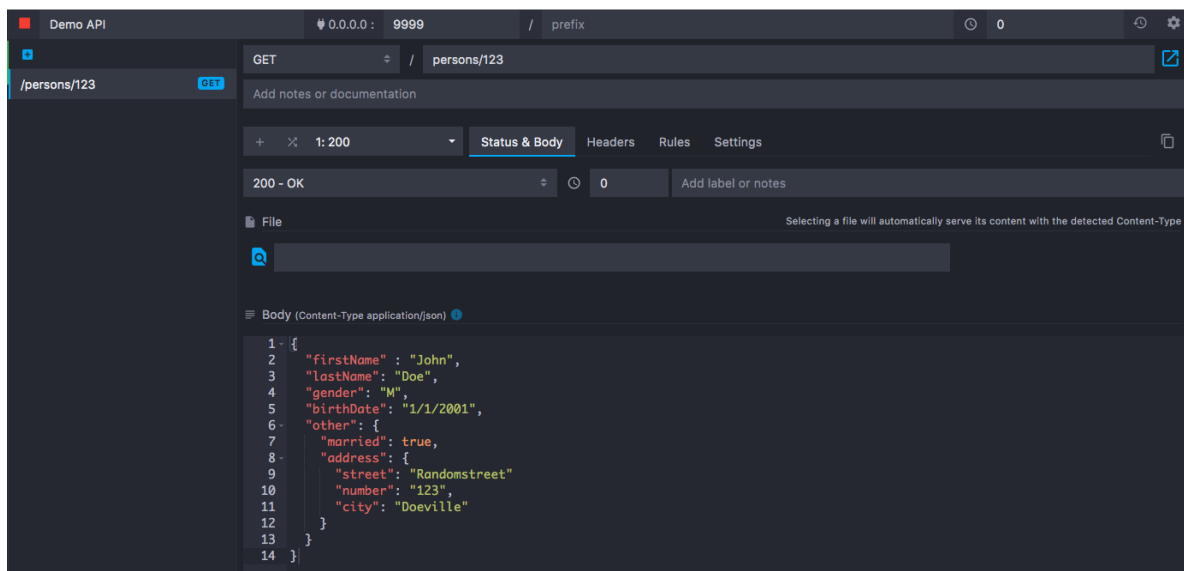
OK

Cancel

Mapping Data Example

Another use case for the *Initialize variables* task is to map data into a structure or format that is more suitable for tasks that follow after it.

Let's assume we want to fetch data from a HTTP REST service and store some parts (but not all) of the response need to be stored as variables. Such an API can be locally mocked by using for example [Mockoon](#):



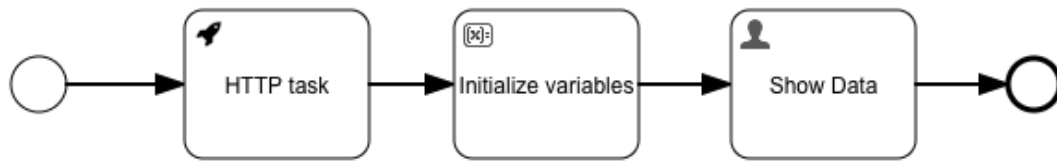
This service response data looks as follows:

```
{  
  "firstName" : "John",  
  "lastName": "Doe",  
  "gender": "M",  
  "birthDate": "1/1/2001",  
}
```



```
"other": {  
  "married": true,  
  "address": {  
    "street": "Randomstreet",  
    "number": "123",  
    "city": "Doeville"  
  }  
}
```

The response returns more data than needed, so we'll use the *Initialize variables* task to filter the data and put it in the format we want. The BPMN process model now looks as follows:



Configure the HTTP Task:

- Request method: *GET*
- Request URL: <http://localhost:9999/persons/123> (same as the mocked api URL)
- Save response as JSON: *checked*
- Save response as a transient variable: *checked*

Details		^
Request method	GET	⚡
Request headers		⛶
Request URL	http://localhost:9999/persons/123	
Request body		📄
Request body encoding		⛶
Request timeout		⚡
Disallow redirects	<input type="checkbox"/>	⚡
Fail status codes		⛶ ⚡
Handle status codes		⛶ ⚡
Ignore exception	<input type="checkbox"/>	⚡
Exception Mappings		⛶
Response variable name	response	
Save request variables	<input type="checkbox"/>	⚡
Save response details	<input type="checkbox"/>	⚡
Result variable prefix		
Save response as JSON	<input checked="" type="checkbox"/>	⚡
Save response as a transient variable	<input checked="" type="checkbox"/>	⚡
Execute parallel in same transaction	<input type="checkbox"/>	














NOTE

The response of the HTTP task is stored as-is as a **transient** variable. This means that the variable will not be persisted at the end of the database transaction (which typically we don't want: the response might be large and storing it would be bad for performance). The Initialize Variables task will instead determine what gets stored and will use the transient JSON variable as input.

In the *Initialize variables* configuration we'll map the JSON response to primitive variables:

- *firstName* gets value type *variable* and value `response.firstName`.

- *lastName* uses an expression `${response.lastName}`. There is no specific reason to not use the *variable* value type here, except for demonstrating the usage of expressions to do the same as with the *variable* type.
- *married* gets value type *variable* and value `response.other.married`, because it's not mapped from the root, but from the *other* nested JSON object.
- *street*, *number* and *city* get value type *variable* and values `response.other.address.street`, `response.other.address.number` and `response.other.address.city`.

Init variables				
	Target	Name	Value Type	Value
 	<input type="text"/>	firstName	Variable	response.firstName
 	<input type="text"/>	lastName		<code>\${response.lastName}</code>
 	<input type="text"/>	married	Variable	response.other.married
 	<input type="text"/>	street	Variable	response.other.address.street
 	<input type="text"/>	number	Variable	response.other.address.number
 	<input type="text"/>	city	Variable	response.other.address.city

[+ Add Variable](#)

Cancel OK

The form in the user task simply displays these variables by binding the fields to the variable values (we're again use one variable per field, similar to the [basic example](#)):

First Name <input type="text" value="{{firstName}}"/>	Last Name <input type="text" value="{{lastName}}"/>	Married <input type="checkbox"/> ...
Street <input type="text" value="{{street}}"/>	Nr. <input type="text" value="{{number}}"/>	City <input type="text" value="{{city}}"/>

Deploy this model and start a new instance. When opening the *Show Data* user task, the data fetched from the HTTP service is now displayed in the form:

Task

Show Data

Created: less than a minute ago
Assignee: Flowable Admin
Due: No due date

Complete Save

Task Conversation Comments People Subtasks Documents History

First Name: John Last Name: Doe ☒ Married

Street: Randomstreet Nr.: 123 City: Doeville

When running with Flowable Inspect enabled, we can see that the HTTP response is not stored as a variable (as we marked it as *transient*) but the variables from the *Initialize variables* task have been stored:

Test Users Inspect Current Test Available Tests Current form

Show Data
TSKc178d76d-4395-11eb-9ac1-acde48001122

Execution tree
Search
☒ InitVariablesProcess
☐ Show Data

Variables
Evaluate expression now

Scope	Name	Type	Value
	number	string	123
	firstName	string	John
	lastName	string	Doe
	city	string	Doeville
	street	string	Randomstreet
	initiator	string	admin
	married	boolean	true

HTTP task Initialize variables Show Data

Advanced

Name expressions

The name column of the *Init variables* configuration can be an expression. For example `${newVariableName}` is valid if `newVariableName` comes from a form field that was set in a user task form or service task before.

JSON Arrays

Using the *JSON Object* value type allows to model JSON variables with a complex structure. The *JSON Array* value type adds the abilities to store lists.

When selecting the *JSON Array* type, optionally the default size of the array can be configured. Default values can be set by using zero-based indexes (for example by using `addresses[0].street`) in the *name* column and providing a value:

Init variables

	Target	Name	Value Type	Value	
<div><div></div><div></div><div></div></div>	<div></div>	<div>customer</div>	<div>JSON Object ▾</div>		<div></div>
<div><div></div><div></div><div></div></div>	<div>customer</div>	<div>addresses</div>	<div>JSON Array ▾</div>	<div>Number of elements:</div> <div>3</div>	<div></div>
<div><div></div><div></div><div></div></div>	<div>customer</div>	<div>emails</div>	<div>JSON Array ▾</div>	<div>Number of elements:</div> <div></div>	<div></div>
<div><div></div><div></div><div></div></div>	<div>customer</div>	<div>addresses[0].street</div>	<div>String ▾</div>	<div>Sesame Street</div>	<div></div>
<div><div></div><div></div><div></div></div>	<div>customer</div>	<div>addresses[1].city</div>	<div>String ▾</div>	<div>New York</div>	<div></div>
<div><div></div><div></div><div></div></div>	<div>customer</div>	<div>name</div>	<div>String ▾</div>	<div>Kermit the Frog</div>	<div></div>

+ Add Variable

Cancel

OK