 **Talend User Components tCamunda\***

# Purpose

This bundle of components is dedicated to work with Camunda BPMN. With the help of these components we can start Camunda BPMN process instances and we can integrate Talend jobs als external task worker into a Camunda BPMN process.

|  |  |
| --- | --- |
| **Component** | **Purpose** |
| tCamundaStartProcessInstance | Starts a process instance |
| tCamundaExtTaskFetchAndLock | Fetch tasks from Camunda and provide them as ongoing flow to the job |
| tCamundaExtTaskComplete | Sends the complete-response to Camunda – the “OK” response |
| tCamundaExtTaskBpmnError | Sends the bpmnError-response – in case of business-related problems occurs |
| tCamundaExtTaskFailure | Sends a failure response – in case of technical errors occors |
| tCamundaExtTaskUpdateVariables | Updates process variables without ending the current tasks |
| tCamundaExtTaskUnlock | Unlock a fetched task in case of the worker cannot finish the work right in time or the work is not possible anymore. |

These components working with the Camunda API – a clear RESTful wenservice.

Camunda provides a complete documentation: <https://docs.camunda.org/manual/7.7/reference/rest/external-task/>

## Talend-Integration

You find these components in the studio in the palette under Camunda.

# Component tCamundaStartProcessInstance



This component starts a Camunda BPMN process instance and can provide variables (aka process variables) as well as return them.

The request returns when the process reaches the first asynchronous point. An asynchronous point can be an external task or a dedicated asynchronous point which means Camunda persists the process variables at this point.

That means it is necessary to keep in mind the duration of one call to start a process instance depends on the design of the process and it is highly recommended to design the process in a way an asynchronous point reach at best very early.

## Basic settings

|  |  |
| --- | --- |
| **Property** | **Content** |
| Use  Alternate Endpoint +  Alternate Endpoint | With this option you can setup the path to the Camunda-API. As long as you run Camunda within a Tomcat you do not need that, if Camunda is running as Spring-Boot-Service, it is necessary. |
| Camunda-Server | The URL of the Camunda server in case of you do not use the alternate endpoint. |
| Camunda-engine | Within a Camunda-application server you can run multiple engines. |
| Need authentication + User + Password | If needed setup here the login credentials. |
| Process definition key | Every deployed process has a key (or name). To address the right process, it is mandatory to name it here. |
| Business Key | The process instance can have a business key. This is a parameter which is available in the whole process. If you start a process for a dedicated object (like a customer, an order etc.) it is a good idea to set their ID as business key.  In the last Camunda versions (I guess since 7.1) the business key is not unique in the database, so you can start a process agin with the same business key. |
| Schema variables | The schema variables will be sent to the process as process variables. By this help of those variables you provide the necessary information for the process what to do. |
| Variables | This table contains for every schema column one entry.   |  |  | | --- | --- | | **Column** | **Meaning** | | Camunda Var Name | You can setup the actual name of the variable within the process. Within Camunda, a variable does not have to fit the Java rules for identifier. E.g. you can add a minus to the name, but now you would not be able to address it with a schema column | | Value | Setup here the value you want to set. E.g. you can use the variables from a tFlowToIterate or context variables | | Value type | Primitive type: String, Boolean, Number will be detected automatically. Date will be String formatted.  JSON formatted value: a value which is a JSON document  XML formatted value: a value which is an XML document | | Java object type | In case of JSON or XML documents you have to specify the Java class which you want to use in Camunda to bind the values. | | Only in Response | If the process variable should not be sent but requested in the response. | |
| Return variables in response | Cause the process to return the sent variables in the response. |

## Advanced settings

|  |  |
| --- | --- |
| **Property** | **Content** |
| Timeout for requests (ms) | The timeout for requests |
| Max retries after failure | The means retries after failures in the communication to the Caumda API.  It has nothing to do with the failure response. |
| Wait ms after error before repeat request | The time in ms to wait until a failed request will be retried |

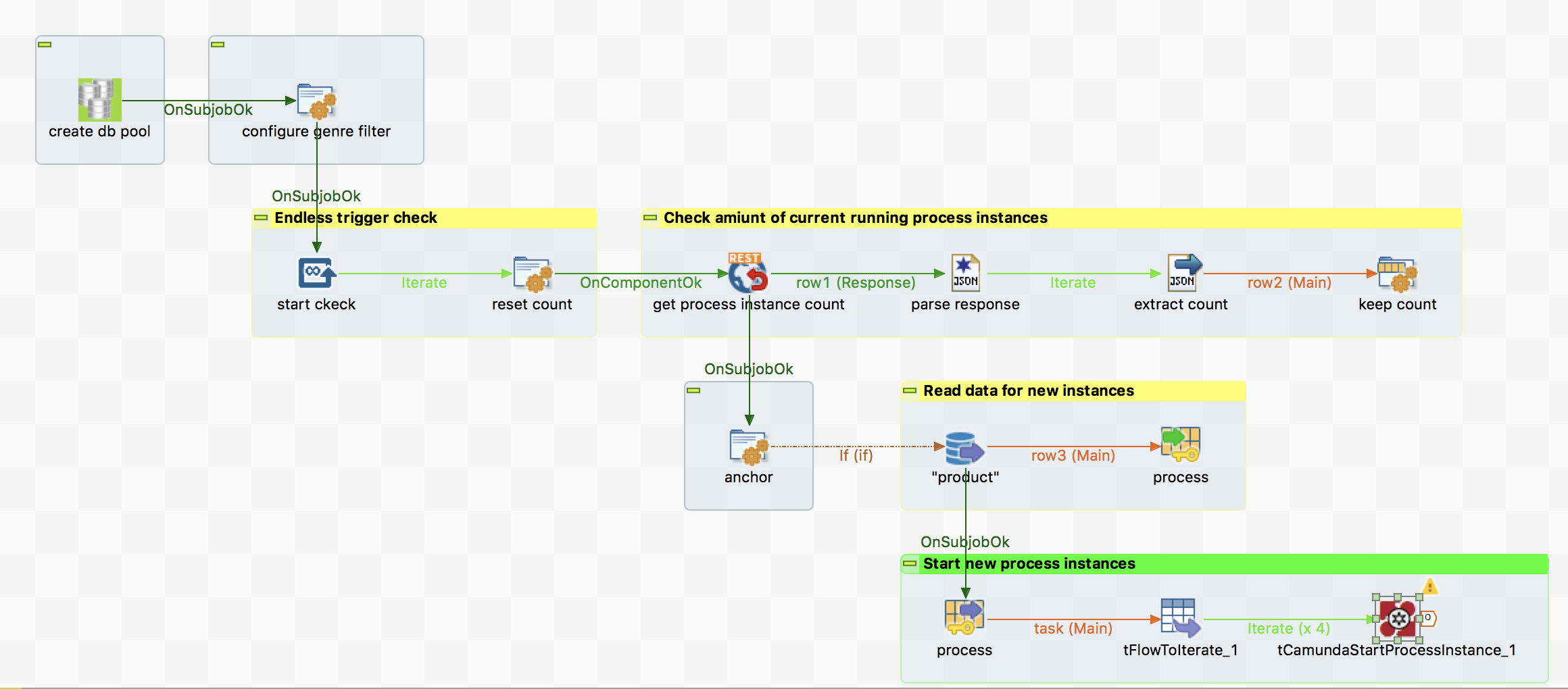
## Return values

|  |  |
| --- | --- |
| **Return value** | **Content** |
| ERROR\_MESSAGE | If the parsing will fail, the error message goes here. |
| NUMBER\_RESPONSE | Number of Responbse flow records (typically 1) |
| PROCESS\_DEFINITION\_KEY | The used process definition key |
| PROCESS\_INSTANCE\_ID | The id of the new created process instance |
| PROCESS\_INSTANCE\_ENDED | true or false if the process instance has been ended already. |
| PROCESS\_INSTANCE\_SUSPENDED | True or false if the started process is suspended |

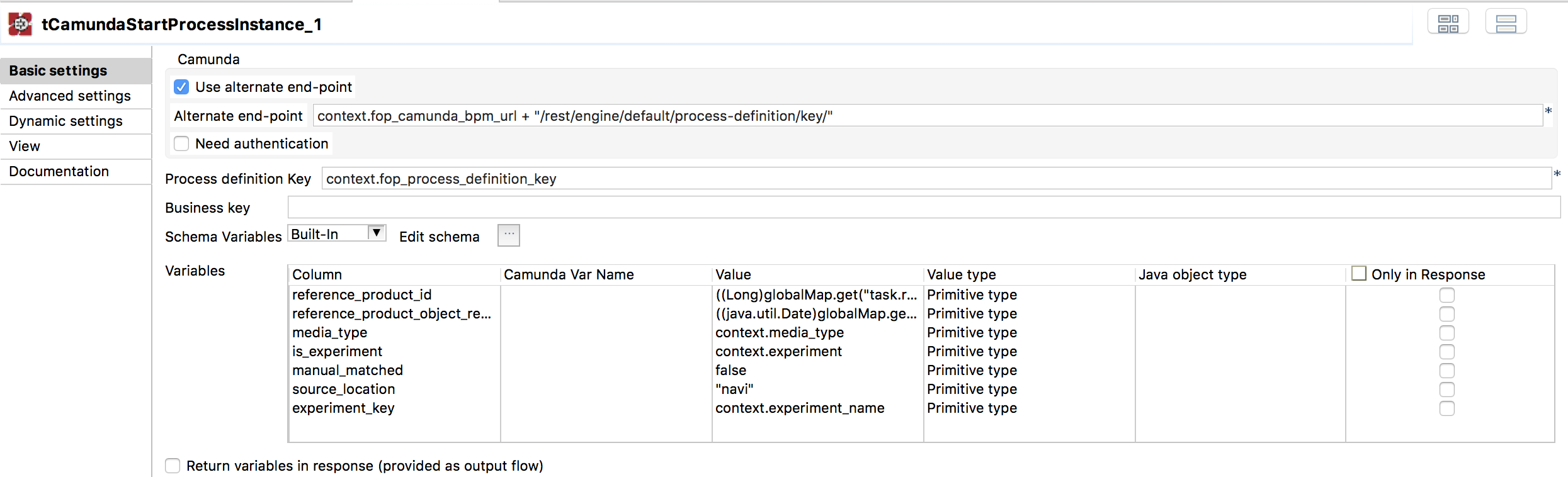
## Scenario

In this scenario the job has to take care we have as much as allowed process instances running – expecting there are enough input data.

The job tries every 10s if the amount of currently running process instances is less than the maximum und starts new instances (difference between current running instances and the maximum).



Here the basic settings of the camunda start component:



# Component tCamundaExtTaskFetchAndLock



This component controls the connection to the Camunda server and is constantly fetching tasks from the topic.

A job containing this component is a external task worker. Tasks will be fetched from the worker and locked for the worker. Only the locking worker can complete the task.

The tasks will be delivered as flow records with the requested process variables as schema columns.

The component frequently sent fetch-requests to the Camunda server. Within one call the component can receive a number of requests at once.

There are 2 modi to run this component.

|  |  |
| --- | --- |
| **Mode** | **Description** |
| Simple mode | Every received task is one record in the flow of the outgoing flow of the component and the requested variables are the schema columns.  The following processing goes task by task. |
| Bulk mode | Activate this with the option “Return all fetched tasks at once as json array”  The component can combine the meta data of the received task and the requested variables to one json object per task and put all received task (within one fetch request) to a json array.  This allows to process all tasks at once.  It is necessary to know every task must have its own response (will be explained later). |

## Basic settings

|  |  |
| --- | --- |
| **Property** | **Content** |
| Use  Alternate Endpoint +  Alternate Endpoint | With this option you can setup the path to the Camunda-API. As long as you run Camunda within a Tomcat you do not need that, if Camunda is running as Spring-Boot-Service, it is necessary. |
| Camunda-Server | The URL of the Camunda server in case of you do not use the alternate endpoint. |
| Camunda-engine | Within a Camunda-application server you can run multiple engines. |
| Need authentication + User + Password | If needed setup here the login credentials. |
| Worker ID | The ID of the worker. Every worker instance must have its own unique ID. Camunda locks a task for a specific worker instance which has fetched the task. |
| Topic name | The name of the topic from wich the tasks have to be fetched.  An external task in Camunda sends its task in an arbitrary topic from which the worker fetches them. Configure the name of the desired topic here. |
| Use priority | Higher prioritized will be fetched first. |
| Max tasks to fetch | Number of maximal tasks to fetch and lock. |
| Task lock duration (ms) | The time a task will be locked for the current worker. Within this time the task must be completed by this worker. If this time will be reached, the task cannot be completed anymore and Camunda put the task back into the topic for a new run. |
| Runtime in seconds or end date | The component continues fetching as long as the runtime or the end date is reached. If there is nothing set here, the job runs without normal end and must be killed to end it. |
| Time between fetches | Seconds or milliseconds between fetchAndLock calls (depending of the option “Milliseconds”).  With this parameter and the max task to fetch parameter you can control the performance. For task running in a short time it is better to fetch them in a shorter time period. |
| Schema Variables | The component can send and receive variables. |
| Return all currently fetched tasks at once as json array node | This option enables the so-called bulk mode. If set to false the component works in the single mode.  With this option the component builds per task one json object with the Camunda task technical attributes and the process variables also as attributes of the json object.  All such json object refelcting a task will be put together in a json array and be assigned to a choosable schema column (see next setting) |
| Column to put in the task array | In the bulk mode visible.  Choose the schema column (must be od type Object) to which the task array will be assigned. |
| Variables | In the single mode visible.   |  |  | | --- | --- | | **Column** | **Meaning** | | Column | Every column will be requested from the process. | | Process variable name | If the variable name differs from the name in the process, put here the real name of the variable | | Allow missing | Check here if the variable can be missed or not. In case a variable is missing the request fails and component throws an error. | |
| Requested variables | In the bulk mode visible.  A list of the variables to be requested. |
| Convert json Strings to plain json values | Camunda transports json formatted values as escaped String. This is inconvenient for the next components to parse the json. It must be transformed into plain json. This option does it for you. |

## Advanced settings

|  |  |
| --- | --- |
| **Property** | **Content** |
| Timeout for requests (ms) | The timeout for requests |
| Max retries after failure | The means retries after failures in the communication to the Caumda API.  It has nothing to do with the failure response. |
| Wait ms after error before repeat request | The time in ms to wait until a failed request will be retried |

These settings are also applied for all response calls in the other response related components.

## Return values

|  |  |
| --- | --- |
| **Return value** | **Content** |
| ERROR\_MESSAGE | If the parsing will fail, the error message goes here. |
| NB\_LINE | The number of outgoing rows. |
| CURRENT\_TASK | Related to the single mode.  The current task as json object including the Camunda task meta attributes and the requested process variables as attributes. |
| CURRENT\_FETCHED\_TASKS\_AS\_ARRAY | Related to the bulk mode.  All tasks as json array of object (see CURRENT\_TASK for one object) |
| CURRENT\_TASK\_ID | Related to the single mode.  The ID of the current task. |
| CURRENT\_PROCESS\_ID | Related to the single mode.  The ID of the current process instance. This is necessary for updating variables of a process. |
| NUMBER\_FETCH\_REQUESTS | Number of fetch requests |
| NUMBER\_FETCH\_REQUESTS\_WITH\_TASKS | Number of fetch requests with which have received tasks. |
| NB\_TASK\_CURRENTLY\_FETCHED | Number of tasks fetched in the last fetch request |
| IS\_LAST\_TASK\_CURRENTLY\_FETCHED | true or false if this task is the last of the currently fetched tasks. |
| NB\_TASKS\_TOTAL | The number of all fetched tasks in total. |

## Delivering the task information

As already mentioned, the task carries Camunda specific attributes and the requested process variables.

### Single mode

Every received task causes a record in the input flow and the schema columns are filled with the values of the requested process variables.

The return value CURRENT\_TASK contains both as json object:

In this example we have 3 requested process variables: productId, schema, update.

They are added to the json object at the end.

Red are the Camunda task attributes and blue the added process variables.

{

"taskId": "1f064dee-b9c6-11e8-ae8e-acde48001122",

"id": "1f064dee-b9c6-11e8-ae8e-acde48001122",

"processInstanceId": "1f0626ce-b9c6-11e8-ae8e-acde48001122",

"lockExpirationTime": "2018-09-16T17:37:01.113+0200",

"activityId": "ServiceTask\_Logger",

"activityInstanceId": "ServiceTask\_Logger:1f064ded-b9c6-11e8-ae8e-acde48001122",

"errorMessage": null,

"errorDetails": null,

"executionId": "1f064dec-b9c6-11e8-ae8e-acde48001122",

"processDefinitionId": "externalTaskCamunda:1:c0238ac8-7f61-11e8-a0c2-acde48001122",

"processDefinitionKey": "externalTaskCamunda",

"tenantId": null,

"retries": null,

"suspended": false,

"workerId": "worker-1",

"priority": 0,

"topicName": "testTopic",

"productId": 112,

"schema": "navi",

"update": {

"testvar": 112

}

}

In the original response from Camunda the “update” variable was send as String value with escaped double quotas.

In the combined task json we see the result of the option “Convert json Strings to plain json values”.

The value is a normal json object.

### Bulk Mode

Every record in the input flow contains a schema column which has assigned all the received tasks as json array.

A flow record will only be created if there are tasks received.

The content of the task array looks like this (Example with maxTasks = 2) :

Note the tasks have different taskIds and different processInstanceIds (bold highlighted).

You will find this json array in the configured schema column and also in the component return value: CURRENT\_FETCHED\_TASKS\_AS\_ARRAY and of course in the schema column which is chosen to transport the json task array.

[

{

"taskId": "a80af550-b9cb-11e8-ae8e-acde48001122",

"id": "**a80af550-b9cb-11e8-ae8e-acde48001122**",

"processInstanceId": "**a80af542-b9cb-11e8-ae8e-acde48001122**",

"lockExpirationTime": "2018-09-16T18:18:33.880+0200",

"activityId": "ServiceTask\_Logger",

"activityInstanceId": "ServiceTask\_Logger:a80af54f-b9cb-11e8-ae8e-acde48001122",

"errorMessage": null,

"errorDetails": null,

"executionId": "a80af54e-b9cb-11e8-ae8e-acde48001122",

"processDefinitionId": "externalTaskCamunda:1:c0238ac8-7f61-11e8-a0c2-acde48001122",

"processDefinitionKey": "externalTaskCamunda",

"tenantId": null,

"retries": null,

"suspended": false,

"workerId": "worker-2",

"priority": 0,

"topicName": "testTopic",

"productId": 29,

"schema": "navi",

"update": {

"testvar": 29

}

},

{

"taskId": "a80ddb90-b9cb-11e8-ae8e-acde48001122",

"id": "**a80ddb90-b9cb-11e8-ae8e-acde48001122**",

"processInstanceId": "**a80ddb82-b9cb-11e8-ae8e-acde48001122**",

"lockExpirationTime": "2018-09-16T18:18:33.881+0200",

"activityId": "ServiceTask\_Logger",

"activityInstanceId": "ServiceTask\_Logger:a80ddb8f-b9cb-11e8-ae8e-acde48001122",

"errorMessage": null,

"errorDetails": null,

"executionId": "a80ddb8e-b9cb-11e8-ae8e-acde48001122",

"processDefinitionId": "externalTaskCamunda:1:c0238ac8-7f61-11e8-a0c2-acde48001122",

"processDefinitionKey": "externalTaskCamunda",

"tenantId": null,

"retries": null,

"suspended": false,

"workerId": "worker-2",

"priority": 0,

"topicName": "testTopic",

"productId": 30,

"schema": "navi",

"update": {

"testvar": 30

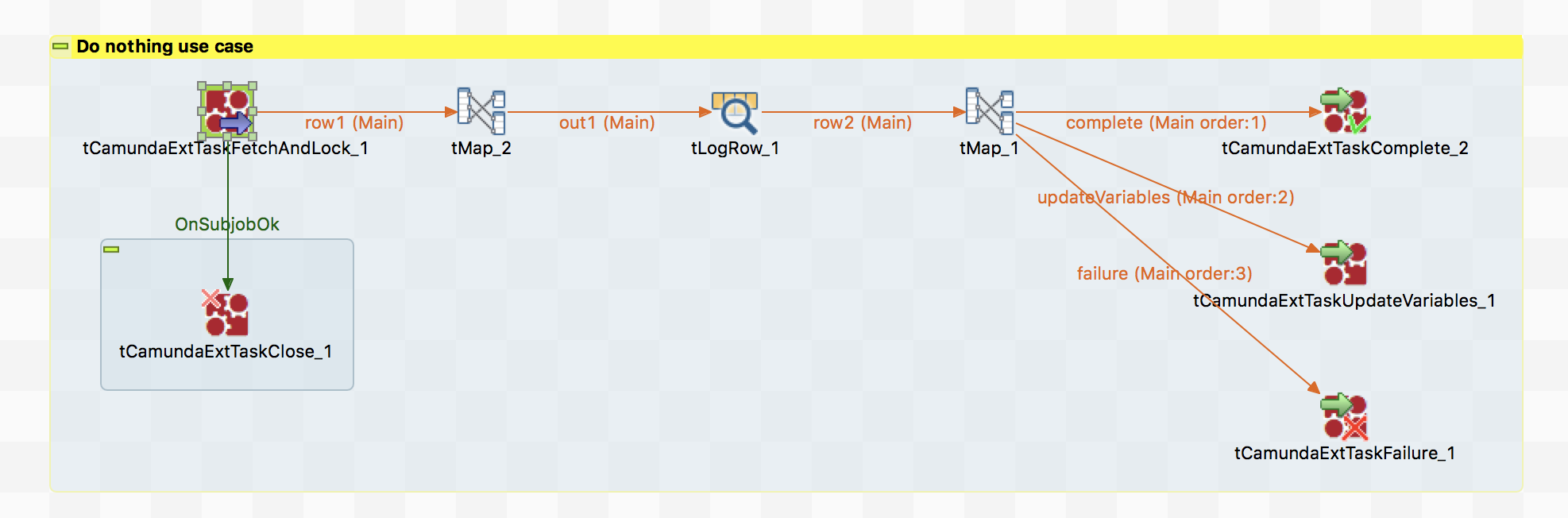
}

}

]

## Scenarios

### Scenario for simple mode



This job actually does not perform real work, it only to demonstrate the principle job structure.

Here the Basic Settings:

