



Talend User Components tJSONDoc*

Purpose

This bundle of components is dedicated to work with JSON documents in the most flexible and unlimited way. Following components exists:

| Component | Purpose |
|----------------|--|
| tJSONDocOpen | Holds the root of the json document and can be initially loaded from various sources |
| tJSONDocInput | Selects objects via JSON-path and reads attribute values |
| tJSONDocOutput | Builds JSON objects or arrays and sets their attributes |
| tJSONDocSave | Renders the final JSON tree pretty formatted as String |

The idea behind these components is to assemble complex JSON documents in a fine grained way. Means you read or write in sub jobs only parts of the documents and the components references its parent nodes and enhance them.

This way you can build or read JSON documents in any way. You can of course read and write similar – means you can perform any transformation.

These components use the JSON path syntax:

<http://goessner.net/articles/JsonPath/>

To check your json path expressions you can use online evaluators like this:

<http://jsonpath.com/>

But for special features like automatically creation of missing nodes in a hierarchy, only the dot-notation is currently supported.

Talend-Integration

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

Component tJSONDocOpen



This component is the root of the JSON document.

This component can create a new empty root (as Object or as Array) or can read the initial JSON document from a source:

- A file
- The input field containing a Java String or plain text representing a JSON document
- A column of an input flow

Because this component carries the necessary library, it is always necessary in any use case.

Where you place this component decides about the overall document structure.

If you place it at the beginning of a job, this means, in your job you build one document.

If you place within a flow (this is always possible) it means you create as much documents as you have rows in your flow (e.g. per request or database record)

Basic settings

| Property | Content |
|----------------------------|--|
| Setup the document as/from | Choose here how to create the initial nodes. |

Various ways to build the initial document

Create an empty ObjectNode

This creates an empty node: {}

Create an empty ArrayNode

This creates an empty array node: []

Read from input flow column

This is especially useful if the document has to be created (initiated) within a flow, e.g. every request of a tRESTRequest have to build its own new document or build for every database record one document. In this case decide in which column of the incoming schema the initial json content has to be read.

Read from file

Point here to a file containing the json content you want to read.

Read from input field below as Java Code

The now visible input field expects Java Code creating the json content as String. This helps in case you need some dynamic in this initial content.

Here an example how it can look like:

```
"{\n"+
+ "  \"level-1\" : {\n"+
+ "    \"level-2\" : [ {\n"+
+ "      \"id\" : \"abc\", \n"+
+ "      \"level-3\" : [ {\n"+
+ "        \"integer-value\" : 10, \n"+
+ "        \"float_val\" : 1.1, \n"+
+ "        \"double_val\" : 1.2, \n"+
+ "        \"bigDec_value\" : 1.3, \n"+
+ "        \"bool_val\" : true, \n"+
+ "        \"date_val\" : \"14-06-2016\", \n"+
+ "        \"jsonString\" : {\n"+
+ "          \"a1\" : \"v1\" \n"+
+ "        }, \n"+
+ "        \"empty_value\" : \"something \n 2 lines\", \n"+
+ "        \"level-4\" : [ {\n"+
+ "          \"integer-value\" : 10, \n"+
+ "          \"float_val\" : 1.1, \n"+
+ "          \"double_val\" : 1.2, \n"+
+ "          \"bigDec_value\" : 1.3, \n"+
+ "          \"bool_val\" : true, \n"+
+ "          \"date_val\" : \"14-06-2016\", \n"+
+ "          \"jsonString\" : {\n"+
+ "            \"a1\" : \"v1\" \n"+
+ "          }, \n"+
+ "        }, {\n"+
+ "          \"integer-value\" : 20, \n"+
+ "          \"float_val\" : 2.1, \n"+
+ "          \"double_val\" : 2.2, \n"+
+ "          \"bigDec_value\" : 2.3, \n"+
+ "          \"string_val\" : \"üöä\", \n"+
+ "          \"bool_val\" : false, \n"+
+ "          \"date_val\" : \"14-06-2016\", \n"+
+ "          \"jsonString\" : {\n"+
+ "            \"a2\" : \"v2\" \n"+
+ "          }, \n"+
+ "        } ] \n"+
+ "      }, {\n"+
+ "        \"integer-value\" : 20, \n"+
+ "        \"float_val\" : 2.1, \n"+
+ "        \"double_val\" : 2.2, \n"+
+ "        \"bigDec_value\" : 2.3, \n"+
+ "        \"string_val\" : \"üöä\", \n"+
+ "        \"bool_val\" : false, \n"+
+ "        \"date_val\" : \"14-06-2016\", \n"+
+ "        \"jsonString\" : {\n"+
+ "          \"a2\" : \"v2\" \n"+
+ "        }, \n"+
+ "      }, {\n"+
+ "        \"integer-value\" : 10, \n"+
+ "        \"float_val\" : 1.1, \n"+
+ "        \"double_val\" : 1.2, \n"+
+ "      } ] \n"+
+ "    } ] \n"+
+ "  } ] \n"+
+ "}
```



```

        "integer-value" : 20,
        "float_val" : 2.1,
        "double_val" : 2.2,
        "bigDec_value" : 2.3,
        "string_val" : "üöä",
        "bool_val" : false,
        "date_val" : "14-06-2016",
        "jsonString" : {
            "a2" : "v2"
        }
    }, {
        "integer-value" : 20,
        "float_val" : 2.1,
        "double_val" : 2.2,
        "bigDec_value" : 2.3,
        "string_val" : "üöä",
        "bool_val" : false,
        "date_val" : "14-06-2016",
        "jsonString" : {
            "a2" : "v2"
        }
    },
    "level-4" : [ {
        "integer-value" : 10,
        "float_val" : 1.1,
        "double_val" : 1.2,
        "bigDec_value" : 1.3,
        "bool_val" : true,
        "date_val" : "14-06-2016",
        "jsonString" : {
            "a1" : "v1"
        }
    } ], {
        "integer-value" : 20,
        "float_val" : 2.1,
        "double_val" : 2.2,
        "bigDec_value" : 2.3,
        "string_val" : "üöä",
        "bool_val" : false,
        "date_val" : "14-06-2016",
        "jsonString" : {
            "a2" : "v2"
        }
    }
    } ]
    } ],
    "example_array" : [ 10, 20 ]
} ]
}
}

```

Return values

| Return value | Content |
|---------------|--|
| ERROR_MESSAGE | If the parsing will fail, the error message goes here. |
| CURRENT_NODE | This is the root JsonNode in this case. |

Component tJSONDocInput



This component is used to read values from the JSON document.

It can build an hierarchy of components (also with tJSONDocOutput) to reflect the JSON document structur.

Basic settings

| Property | Content |
|---|---|
| Parent JSON Document | Choose here the JSONDoc* component which current processing node should be the starting point to read. |
| Take the values from the referenced parent object | If you want to read the json object from the parent component (because of other attribute set etc.) check this option. |
| JSON path to loop | <p>This is the path to the json document which surfs as loop element. You can set here an absolute json-path or a relative attribute path.</p> <p>If the path starts with an \$ the path will be parsed with the original json-path methodology. If the path does not start with an \$, it means it is simple a chain of attributes describing the way to the target loop node but starting from a particular node. This goes typically with a Parent JSON document like tJSONDocInput or tJSONDocOutput.</p> <p>The value "." (standalone) means, the source for the attributes is the referenced parent object. This way 2 components can read from the same object but e.g. different attributes with different constraints (not null or missing). You could also simply check the option above!</p> <p>If the addressed node is an object node, the component reads the attributes from this node and provides only one row.</p> <p>If the addressed node is an array node which the component reads the attributes of the addressed child nodes and provides one row per child node.</p> <p>If the addressed node is a value array, the component provides per element in the array a new row and set the value into the given schema columns. Actually it does not make sense to have a schema with more than one column.</p> |
| Die if attribute does not exists | If the JSON path does not exists and the path is mandatory you can let the component die with a meaningful error message. |
| Attributes | <p>Configure here the attributes you want to read from the json objects.</p> <p>Column: the schema column</p> <p>Alternative Name: You can set here the name of the attribute if it is not the same as the schema column name. This way you can set names which are not compatible with Java conventions like attribute names with a minus e.g.</p> <p>Use Column: Decide here which column you want to fill from the json object.</p> <p>Allow missing: Set this option if the attribute can be missed. The default option is off.</p> <p>Value if attribute is missing: Set here a replacement value for missing attributes. This value could be used later in the job to detect if the attribute was null or missing.</p> |
| Schema | <p>The default schema editor. Please use the Date pattern to parse the Date strings in the json document. JSON actually does not have a standard date pattern or even such a data type. It depends on string parsing to work with dates and timestamps.</p> <p>The component uses an internal default pattern "yyyy-MM-dd'T'HH:mm:ss:SSS" if you do not provide a date pattern in the schema.</p> |

Return values

| Return value | Content |
|---------------|--|
| ERROR_MESSAGE | If the parsing will fail, the error message goes here. |
| CURRENT_NODE | This is the current JsonNode from which the attributes are currently read. |
| NB_LINE | The number of outgoing rows. |

Scenario: Reading a document with multiple levels:

This shows how to chain the processing with Iterate flows.

tJSONDocInput_2

Parent JSON Document: tJSONDocOpen_1 *

JSON path to loop: "\$.participation.rolesnames" * ☐ Die if attribute does not exists

Attributes

| Column | Alternative Name | <input type="checkbox"/> Use column | <input type="checkbox"/> Allow missing | Value if the attribute is missing |
|-------------|------------------|-------------------------------------|--|-----------------------------------|
| rolename | | <input checked="" type="checkbox"/> | <input type="checkbox"/> | |
| language_id | | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | 9999 |

Schema: Built-In Edit schema

The json path points to the node which is the loop element.

Because we have a json path starting with \$ we read just from the root element. But it could also be possible to reference the tJSONDocInput_1 and use a relative path which takes the current node from tJSONDocInput_1 as starting point to find the node(s) to read from.

Take note of the value 9999 in the column Value if attribute is missing. This value will be sent if the attribute is missing at all. A null value is NOT a missing attribute!

Component tJSONDocOutput



This component is dedicated to create and write all kind of json nodes.

It can be chained with other tJSONDoc components and work relatively on top of the current node of the addressed parent component.

Basic settings

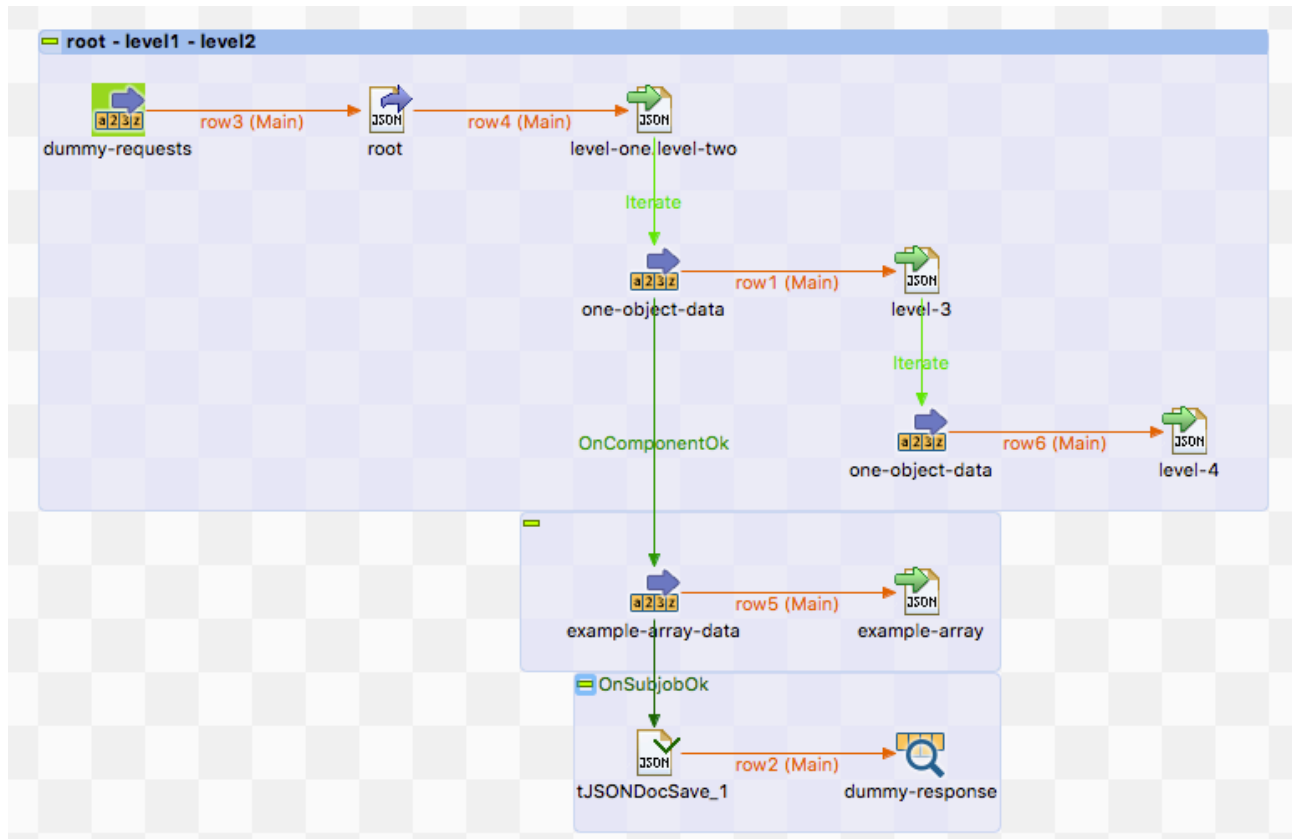
| Property | Content |
|----------------------------------|---|
| Parent JSON Document | Choose here the JSONDoc* component which current processing node should be the starting point to read. |
| JSON path for the current parent | This is the path (in dot-notation) to the current element you want to create and write. The last part of the path is the attribute under which the nodes are created or be written. |
| Output structure | Setup here which kind of output structure you want: Array of Object nodes: The component creates or uses an ArrayNode and add to this array node for every incoming row a new object node and set their attributes. One single object node: The component creates or uses a single ObjectNode and set its attributes. In this mode multiple incoming rows actually does not make sense. The last record will determine the content. Array of values: The component creates a value array and takes every incoming row as one value element. If the schema has multiple schema columns all column values are written as their own array value. It is actually more meaningful to have only one schema column. |
| Attributes | Configure here the attributes you want to read from the json objects. Column: the schema column Alternative Name: You can set here the name of the attribute if it is not the same as the schema column name. This way you can set names which are not compatible with Java conventions like attribute names with a minus e.g. Use Column: Decide here which column you want to fill from the json object. Is a JSON object/array: Check this option if the content of the schema column is a JSON node, otherwise the content will be treated as value and the content will be escaped to be compatible with json strings. If the schema column is of an Object type the content will be taken as JsonNode object, if the content is of String type, the content will be parsed to a JsonNode. Omit attribute if value is null: With this option you can prevent writing the attribute if the value is null |
| Schema | The default schema editor. Please use the Date pattern to parse the Date strings in the json document. JSON actually does not have a standard date pattern or even such a data type. It depends on string parsing to work with dates and timestamps. The component use a internal default pattern "yyyy-MM-dd'T'HH:mm:ss:SSS" if you do not provide a date pattern in the schema. |

Return values

| Return value | Content |
|---------------|---|
| ERROR_MESSAGE | If the parsing will fail, the error message goes here. |
| CURRENT_NODE | This is the current JsonNode from which the attributes are currently written. |
| NB_LINE | The number of incoming rows |

Scenario 1: Write a complex json document

This scenario shows how to build a multi-level json document like the example in tJSONDocOpen.



We have a json document with 4 levels and an additional value array.

Please note the order in which the rows and iterates will be processed.

At first one row will be processed and right after this one row the iteration takes place also once a time.

One iteration per one output flow record. This is important because in the deeper levels you can build json objects as children of the current written object of the addressed parent object.

Here the settings of level-3:

level-3(tJSONDocOutput_1)

Parent JSON Document: tJSONDocOutput_2 - level-one.level-two

JSON path for the parent: "level-3"

Output structure: Array of object nodes

Attributes

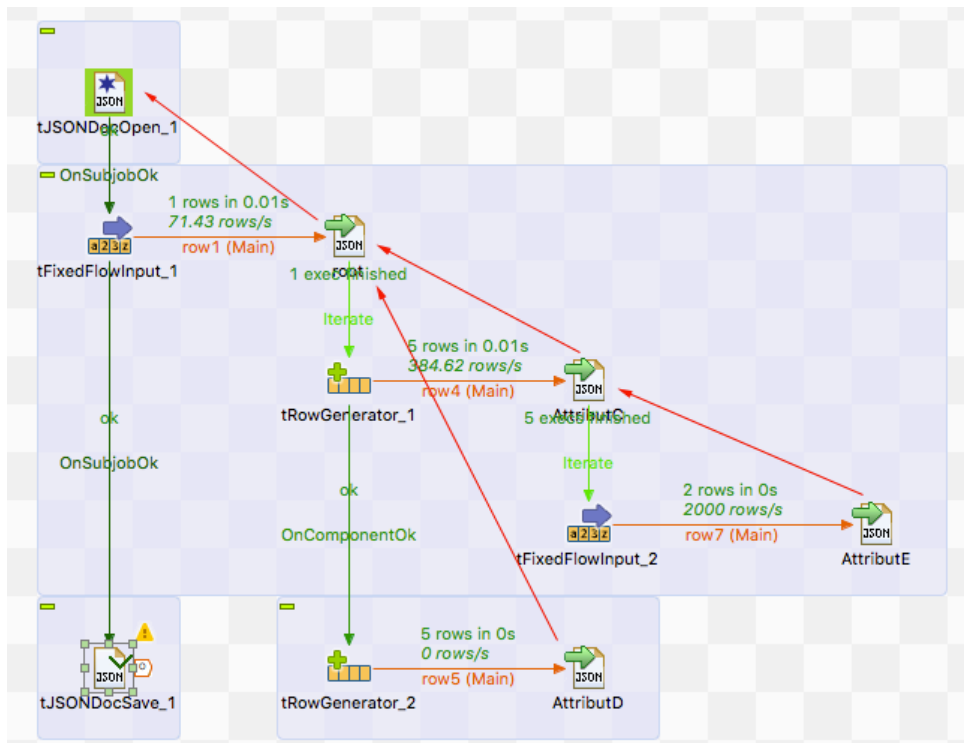
| Column | Alternative Name | Use column | Is a JSON object/array | Omit attribute if value is null |
|--------------|------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| int_val | "integer-value" | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> |
| float_val | | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| double_val | | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| bigDec_value | | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| string_val | | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| bool_val | | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| date_val | | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| jsonString | | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| empty_value | | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

Schema: Built-In Edit schema Sync columns

As you can see it is based on the level-one.level-two node.

Scenario 2: Example of multi-level document creation

This example shows a bit deeper how the concept of referencing to a parent component works.



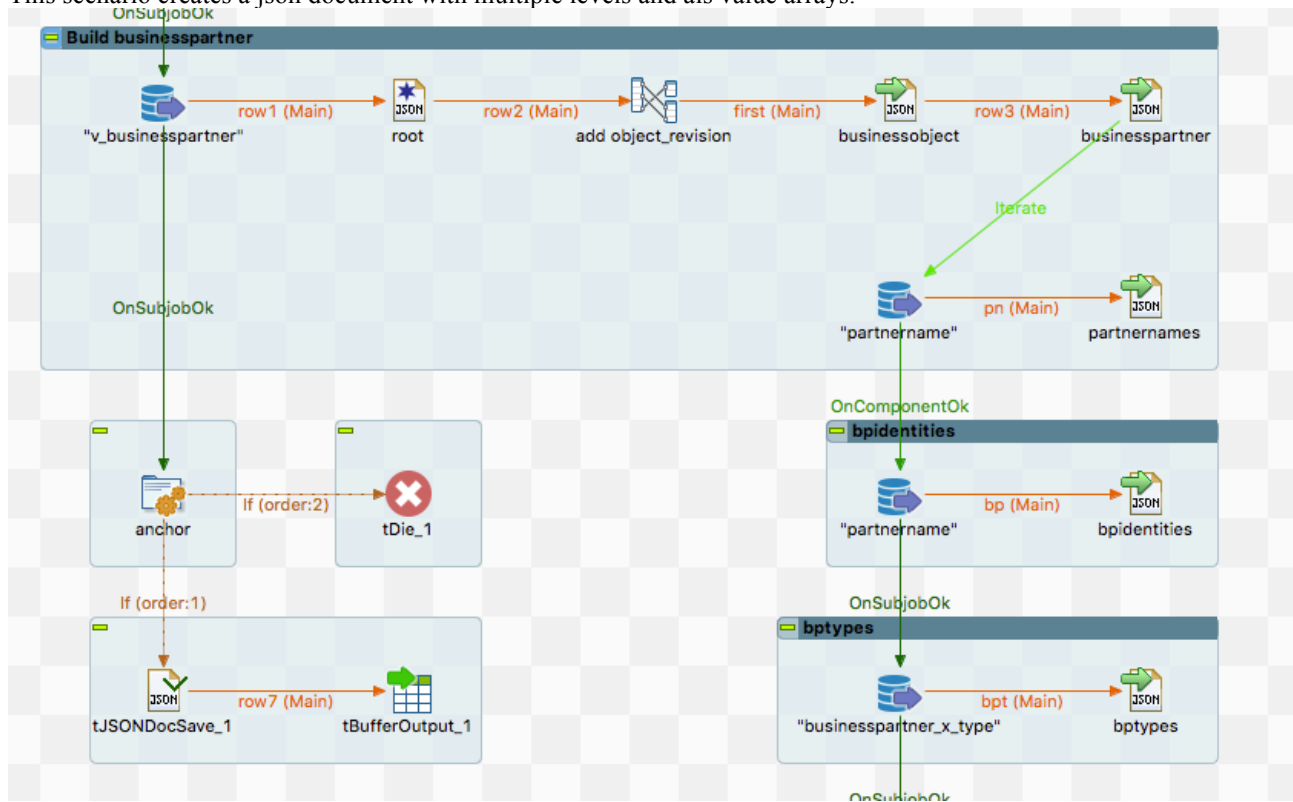
The red arrows show which parent component all components references.

Here the result:

```
{
  "AttributA" : "AAA",
  "AttributB" : "BBB",
  "AttributC" : [ {
    "AttributC_ID" : 1,
    "AttributeE" : [ "EEE1", "EEE2" ]
  }, {
    "AttributC_ID" : 2,
    "AttributeE" : [ "EEE1", "EEE2" ]
  }, {
    "AttributC_ID" : 3,
    "AttributeE" : [ "EEE1", "EEE2" ]
  }, {
    "AttributC_ID" : 4,
    "AttributeE" : [ "EEE1", "EEE2" ]
  }, {
    "AttributC_ID" : 5,
    "AttributeE" : [ "EEE1", "EEE2" ]
  } ],
  "AttributD" : [ {
    "id" : 6
  }, {
    "id" : 7
  }, {
    "id" : 8
  }, {
    "id" : 9
  }, {
    "id" : 10
  } ]
}
```

Scenario 3: A real live scenario to create a complex json document

This scenario creates a json document with multiple levels and als value arrays.



Per record from the database we create a json document and add in the first place the values from the first database input and continue per "businesspartner" with adding more objects like "partnernames".

At the end we write the content with the tJSONDocSave component (see next chapter) in tBufferOutput (we use this job within other jobs and tBufferOutput is a great way to provide content to the parent job).

Component tJSONDocSave



This component is dedicated to provide the JSON document as pretty formatted string to any kind of output. Actually it is not mandatory to use it because it would simple be fine to use the return value `CURRENT_NODE` from `tJSONDocOpen` to have the content of the json document.

Basic settings

| Property | Content |
|-------------------------------|---|
| JSON Document root | Choose here thet JSONDocOpen component which current processing node should be used to render the document output. |
| Write content into a file | With this option and the file chooser setup the output file in which the conent will be written. The charset is UTF-8 and it uses UNIX style line breaks. |
| Output schema column | If there is an output flow, choose here the column in which the content have to set as value. As column type String is needed. |
| Pretty Print | The string output will be formatted in a human readable way. Otherwise everything is in a condensed one line String. |
| Ignore empty values | Empty values and arrays will be filtered out |
| Write content to standard out | Does what is says. This is actually I kind of debug option to see the content on the standard output of the job. |

Return values

| Return value | Content |
|------------------|---|
| ERROR_MESSAGE | If something went wrong, e.g. we cannot write into the file, the error message goes here. |
| OUTPUT_FILE_PATH | The path to the output file if set. This is useful if the path is calculated and you need that in the further processing. |
| JSON_STRING | The String content of the json document. |