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
tJSONDocInputStream	Reads large JSON files and uses a stream parser to read the values or objects.

The idea behind these components is to assemble complex JSON documents in a fain 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 intial 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 especcially 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 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"
    + "        \"jsonString\" : {\n"
    + "          \"a1\" : \"v1\"\n"
. . .
    + "}"
```

The option: "Simplified line breaks" means you can put here content in the way you usually do e.g. in the database input components. In this case the line breaks will be added automatically and you do not need to chain the content with Java String operation and you do not need to quote every line.
Here an example of the simplified notation:

```
"{
   \"level-1\" : {
     \"level-2\" : [ {
       \"id\" : \"abc\",
       \"level-3\" : [ {
         \"integer-value\" : 10,
         \"float_val\" : 1.1,
         \"double_val\" : 1.2
       } ]
      ] }
   }
}"
```

You have to escape the double quotas within the content and it needs a double quotas at the start and the end.
This is also the place where you put the context variable containing your json content.
E.g.:
`context.json`

Read from input field below as plain text
The same content as abow but now as real plain json content without any Java language parts.
This is also a very good help while testing your job. Simply pleace here your test document if you have to parse it.

```
{
  "level-1" : {
    "level-2" : [ {
      "id" : "abc",
      "level-3" : [ {
        "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"
        },
```

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 hierachy of components (also with tJSONDocOutput) to reflect the JSON document structur.

Basic settings

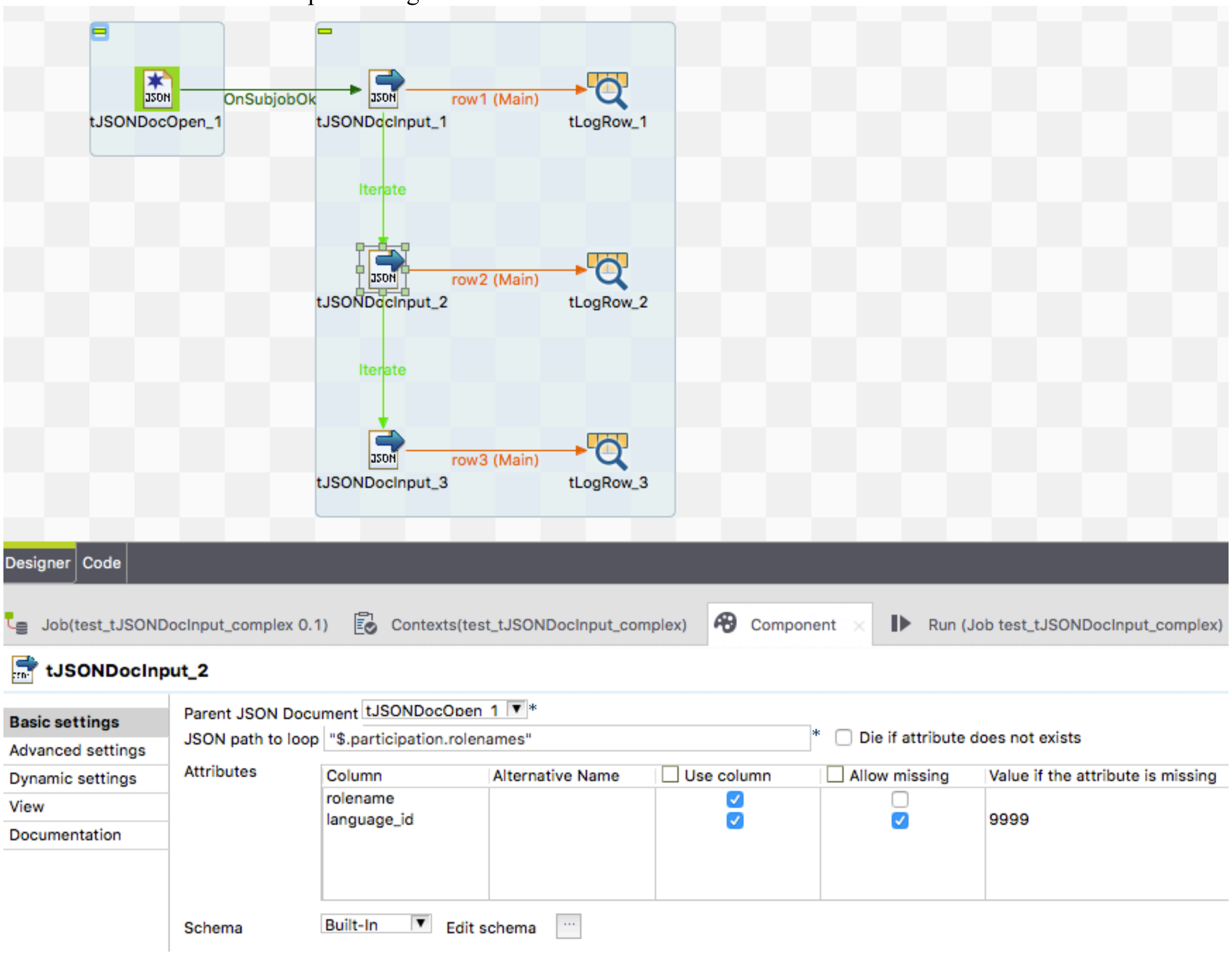
Property	**Content**
Parent JSON Document	Choose here thet JSONDoc* component which current processing node should be the starting point to read.
Take the values from the referenced parent object	If you want to read the json object from the parent component (because of other attribute set etc.) check this option.
JSON path to loop	This is the path to the json document which surfes as loop element. You can set here an absolute json-path or a relative attribute path. **If the path starts with an $ the path will be parsed with the original json-path methology**. **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. **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! **If the addressed node is an object node,** the component reads the attributes from this node and provides only one row. **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. **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.
Die if attribute does not exists	If the JSON path does not exists an the path is mandatory you can let the component die with a meaningful error message.
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. If this expression results in an empty or null name, the schema name will be used instead. **Use Column:** Decide here which column you want to fill from the json object. **Allow missing:** Set this option if the attribute can be missed. The default option is off. **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.
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 read.
NB_LINE	The number of outgoing rows.
NB_NOT_NULL_ATTRIBUTES	The number of attributes which are not missing and not null in the current node
CURRENT_PATH	The JSON-Path to the current node as String

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.rolenames" * ☐ Die if attribute does not exists

Attributes

Column	Alternative Name	☐ Use column	☐ Allow missing	Value if the attribute is missing
rolename		✓	☐	
language_id		✓	✓	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 is 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 send if the attribute is missing at all. A null value is NOT a missing attribute!

Scenario: Reading document with multiple nested arrays

The goal is to have values from the higher levels and the details of the lowest levels in one flow.
There are 2 multile ways: This scenario describes the way by addressing the objects.

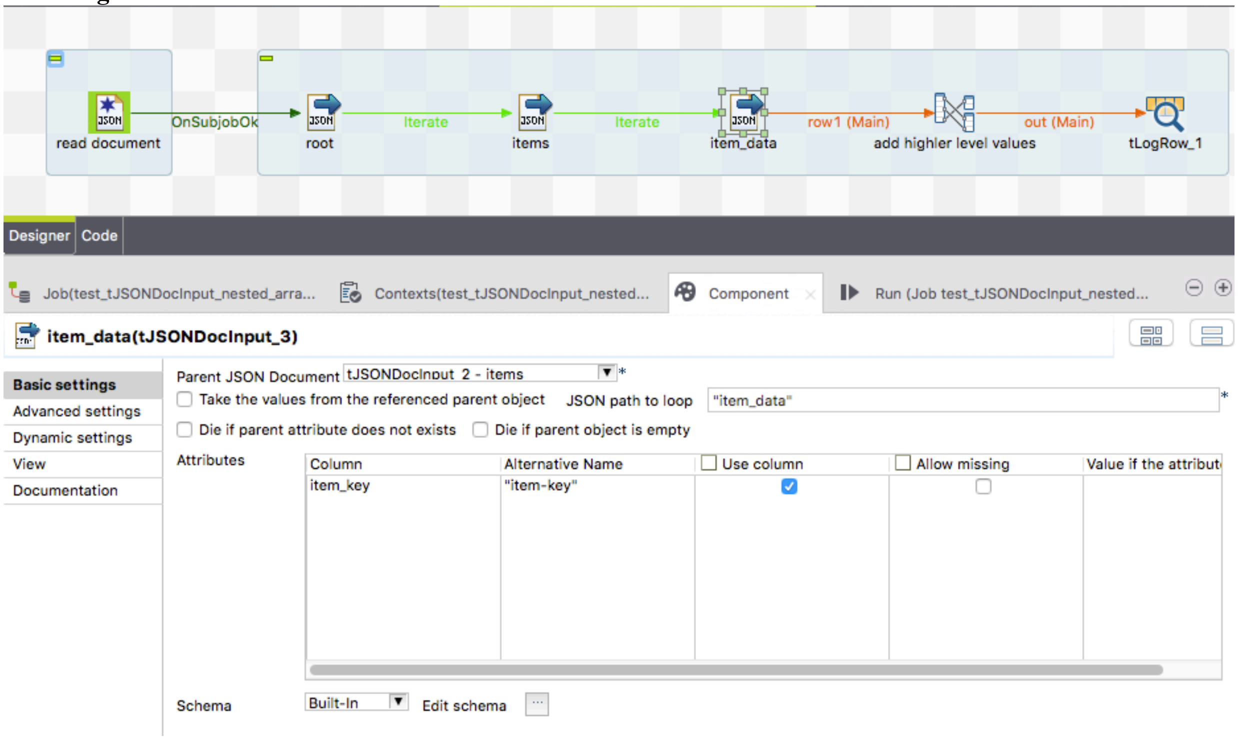
Here the input:

```
[
{
    "header": "global header1",
    "items": [
        {"group_header": "group_header1",
         "item_data": [
               {"item-key": 1},
               {"item-key": 2},
               {"item-key": 3}
         ]
        },
        {"group_header": "group_header2",
         "item_data": [
               {"item-key": 1},
               {"item-key": 2},
               {"item-key": 3}
         ]
        }
    ]
},
{
    "header": "global header2",
    "items": [
        {"group_header": "group_header1",
         "item_data": [
               {"item-key": 1},
               {"item-key": 2},
               {"item-key": 3}
         ]
        },
       {"group_header": "group_header2",
         "item_data": [
               {"item-key": 1},
               {"item-key": 2},
               {"item-key": 3}
         ]
        }
    ]
}
]
```

… and here the desired output:

```
header        |group_header |item_key
-------------------------------------------------------------
global header1|group_header1|1
global header1|group_header1|2
global header1|group_header1|3
global header1|group_header2|1
global header1|group_header2|2
global header1|group_header2|3
global header2|group_header1|1
global header2|group_header1|2
global header2|group_header1|3
global header2|group_header2|1
global header2|group_header2|2
global header2|group_header2|3
```

Job design to solve this use case:



item_data(tJSONDocInput_3)

Parent JSON Document: tJSONDocInput_2 - items *

☐ Take the values from the referenced parent object   JSON path to loop: "item_data" *

☐ Die if parent attribute does not exists   ☐ Die if parent object is empty

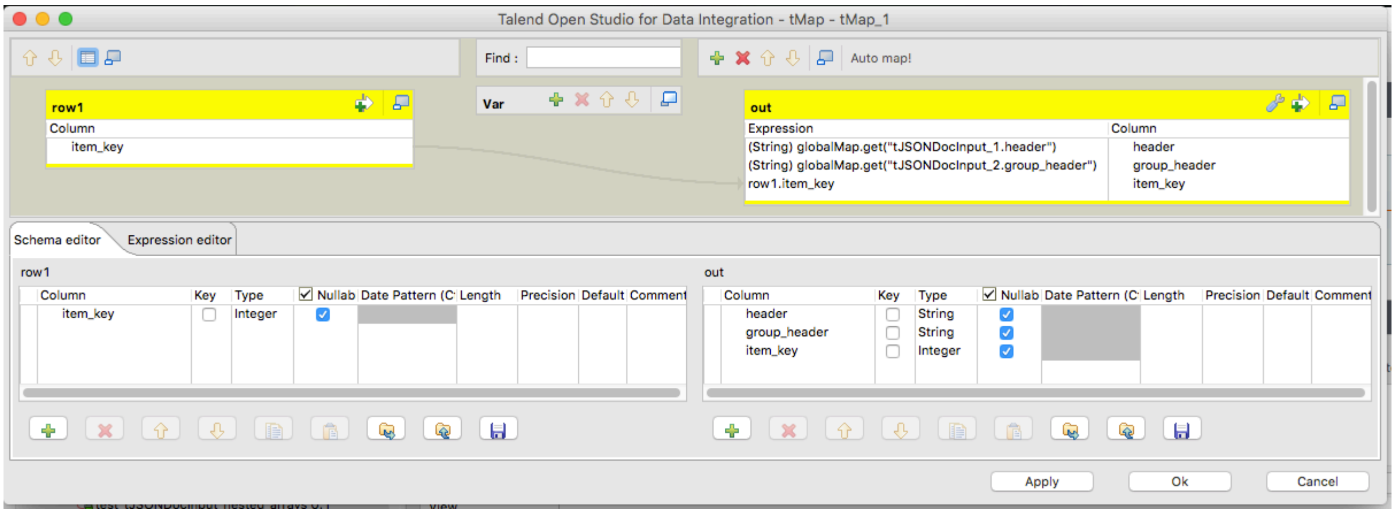
Attributes

Column	Alternative Name	☐ Use column	☐ Allow missing	Value if the attribut
item_key	"item-key"	✓	☐	

Schema: Built-In   Edit schema

The first tJSONDocInput references as parent the tJSONDocOpen component. All other references the current predecessor.

To provide the higher level values we have to use a tMap and adds these values into the output flow.
All components put their current values into the global map with the key: <unique-component-id>.<schema-column-name>
Example refer to the picture below.


Talend Open Studio for Data Integration - tMap - tMap_1

row1

Column
item_key

out

Expression	Column
(String) globalMap.get("tJSONDocInput_1.header")	header
(String) globalMap.get("tJSONDocInput_2.group_header")	group_header
row1.item_key	item_key

row1

Column	Key	Type	Nullab	Date Pattern (C	Length	Precision	Default	Comment
item_key	☐	Integer	✓					

out

Column	Key	Type	Nullab	Date Pattern (C	Length	Precision	Default	Comment
header	☐	String	✓					
group_header	☐	String	✓					
item_key	☐	Integer	✓					

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

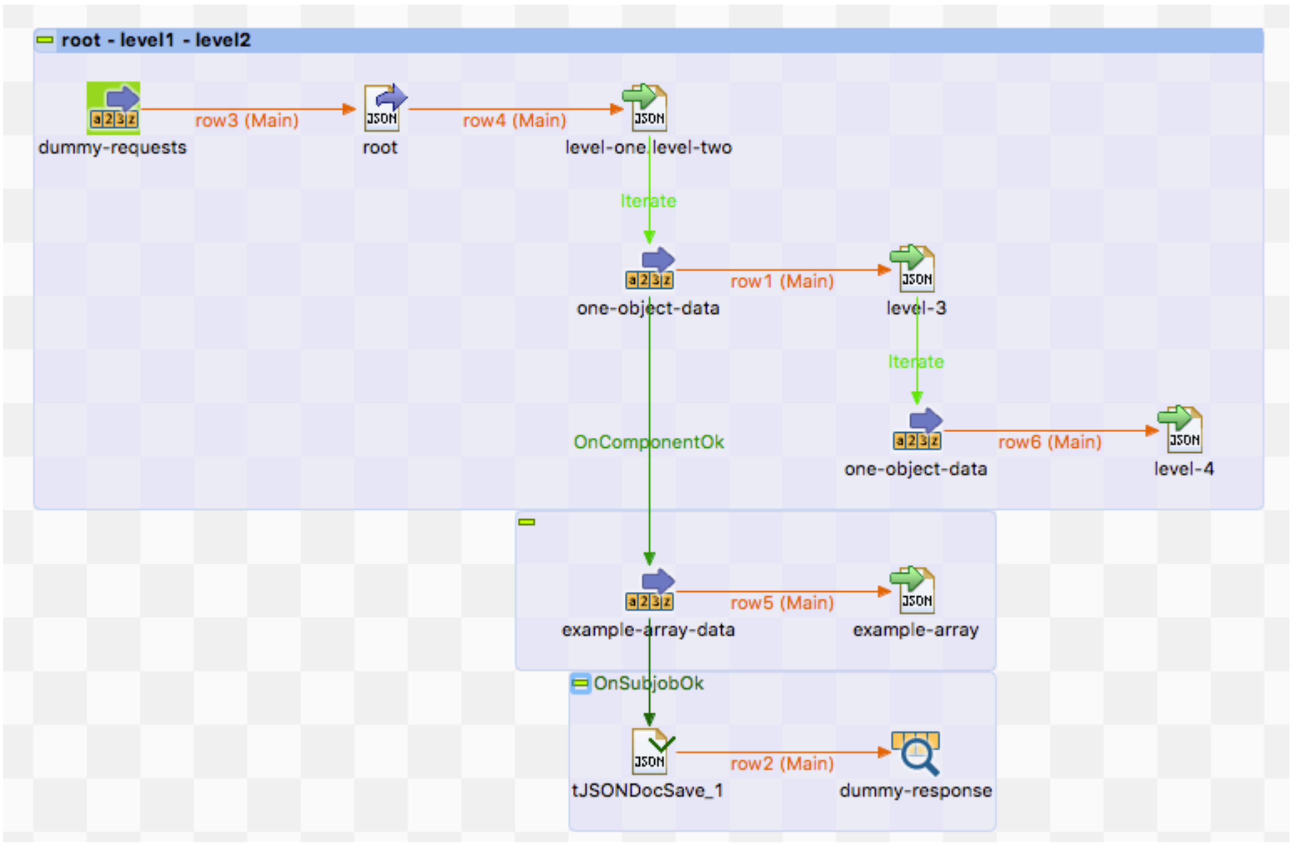
Property	**Content**
Parent JSON Document	Choose here thet JSONDoc* component which current processing node should be the starting point to read.
JSON path for the current parent	This is the path (in dot-notation) to the current element you want to create and write. The last part of the path is the attribute 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. If this expression results in an empty or null name, the schema name will be used instead. **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
CURRENT_PATH	The path to the current node as JSON-Path String

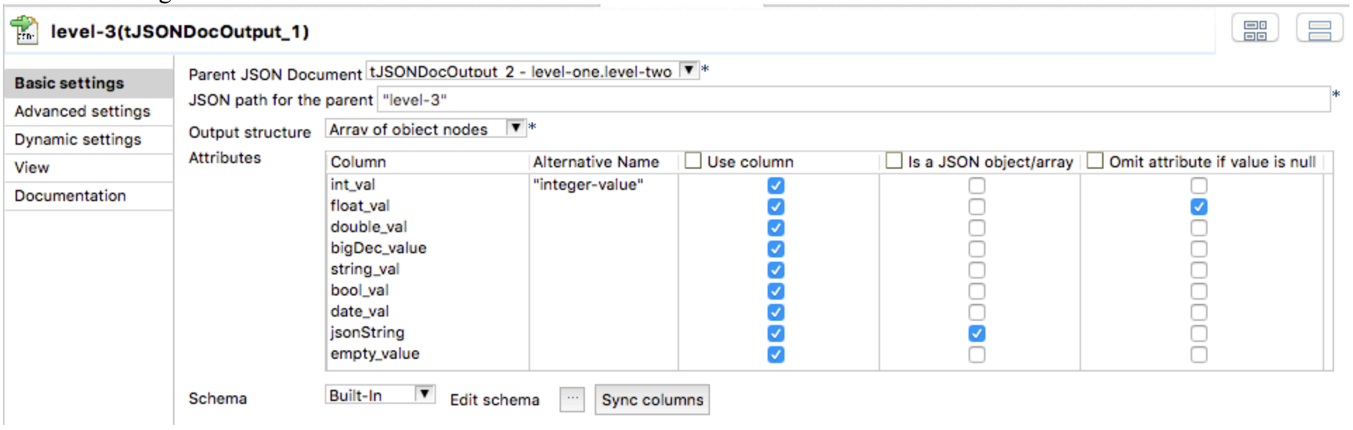
Scenario 1: Write a complex json document

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 processed and right after this one row the iteration takes places 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)

Basic settings | Advanced settings | Dynamic settings | View | Documentation

Parent JSON Document: tJSONDocOutput_2 - level-one.level-two
JSON path for the parent: "level-3"
Output structure: Array of object nodes
Attributes:

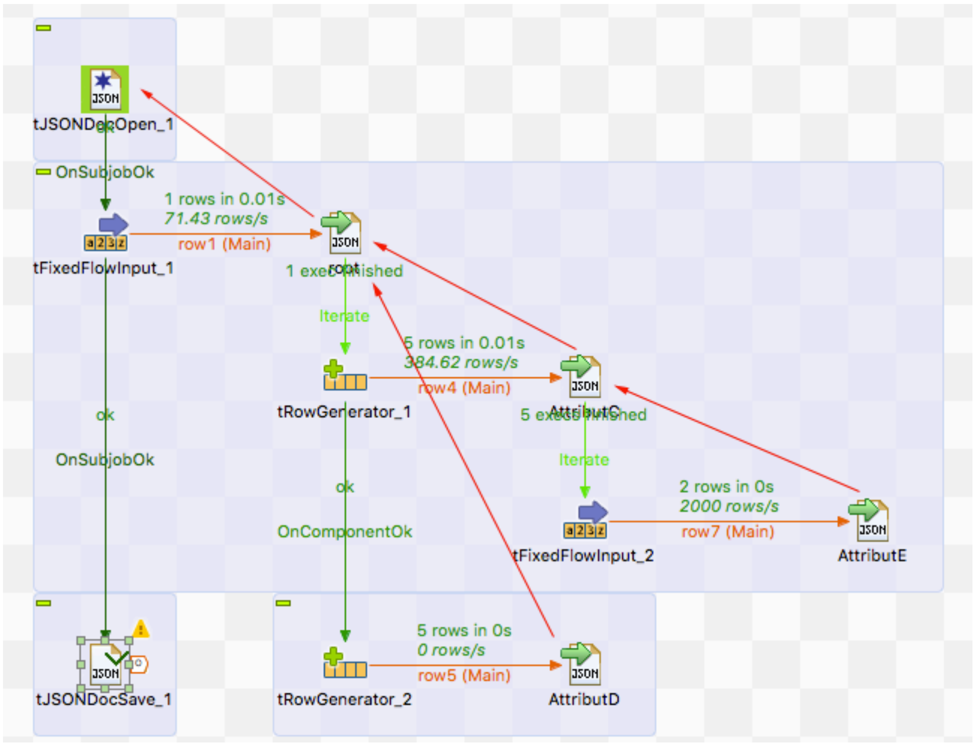
Column	Alternative Name	Use column	Is a JSON object/array	Omit attribute if value is null
int_val	"integer-value"	✓		
float_val		✓		✓
double_val		✓		
bigDec_value		✓		
string_val		✓		
bool_val		✓		
date_val		✓		
jsonString		✓	✓	
empty_value		✓		

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.

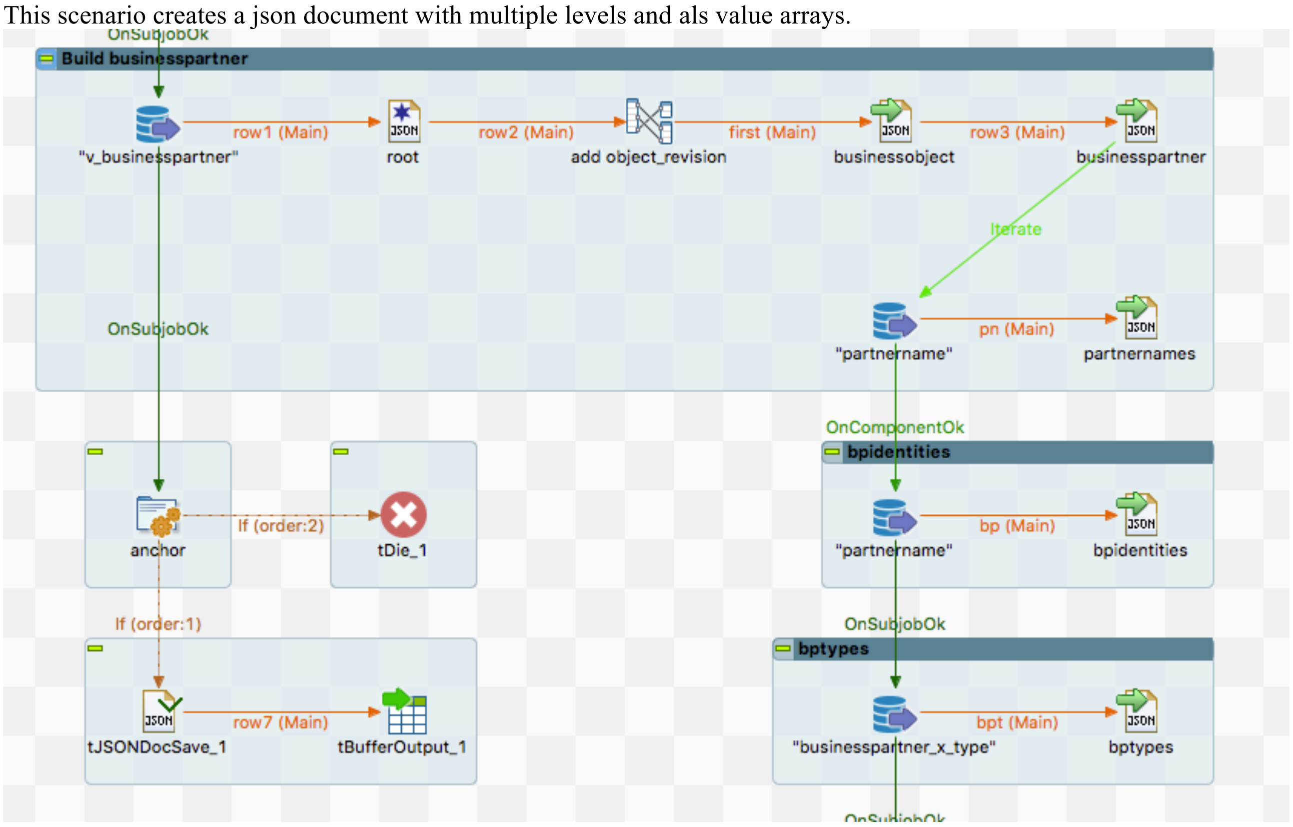


The red arrows show which parent component all components references.
Here the result:

```
{
  "AttributA" : "AAA",
  "AttributB" : "BBB",
  "AttributC" : [ {
    "AttributC_ID" : 1,
    "AttributE" : [ "EEE1", "EEE2" ]
  }, {
    "AttributC_ID" : 2,
    "AttributE" : [ "EEE1", "EEE2" ]
  }, {
    "AttributC_ID" : 3,
    "AttributE" : [ "EEE1", "EEE2" ]
  }, {
    "AttributC_ID" : 4,
    "AttributE" : [ "EEE1", "EEE2" ]
  }, {
    "AttributC_ID" : 5,
    "AttributE" : [ "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 has 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.

Component tJSONDocInputStream

This component is dedicated to read very large JSON files and extract attributes with JSONPath expressions.
In the current state the capabilities of using JSONPath is limited:
Only the dot-notation is allowed
Search is not possible, only addressing of objects and arrays with the notation [*] is possible.

Basic settings

Property	**Content**
JSON file path	Set here the file path of the input file
JSON Path to loop	Address the JSON structure you want to loop over. E.g. address the attribute (referencing an array). The syntax here is not fully JSON Path complient and currently very limited. A path must start with $ and all attributes are separated with . (dot) Arrays must be declared in the path with [*] – currently there is not possibility to access one particular element of an array.
Attributes	**Columns:** Schema column **JSON Path:** it the path starts without $ it will extend the loop path to address the attribute **Use column:** switch off the column if you do not want to set this attribute
Schema	The schema editor. Configure here the schema for reading.
Die on error	Let the component die if something went wrong.
Reject schema	The schema of the reject flow. The reject flow will only be used if you switch off the Die on error option.

Advanced settings

Property	**Content**
Provide JSON loop object	If you want to use the json object returned in every iteration of the loop for some other tJSONDocInput components, switch on this option. Because of the json object must be created and this could have performance impacts, this is an optional feature.

Return values

Return value	**Content**
ERROR_MESSAGE	If something went wrong - the error message goes here.
NB_LINES	Number rows
NB_LINES_REJECTED	Number lines rejected
CURRENT_NODE	This is the current JsonNode referenced by the loop path. This JsonNode is only prsent if you have switched on the advanced option " Provide JSON loop object". To build such an object reduces the performance typically for about 2-5% - depending of the complexity of the loop element.

Scenario: Reading a json with nested arrays

This describes the reading of a small document to illustrate the parsing feature.

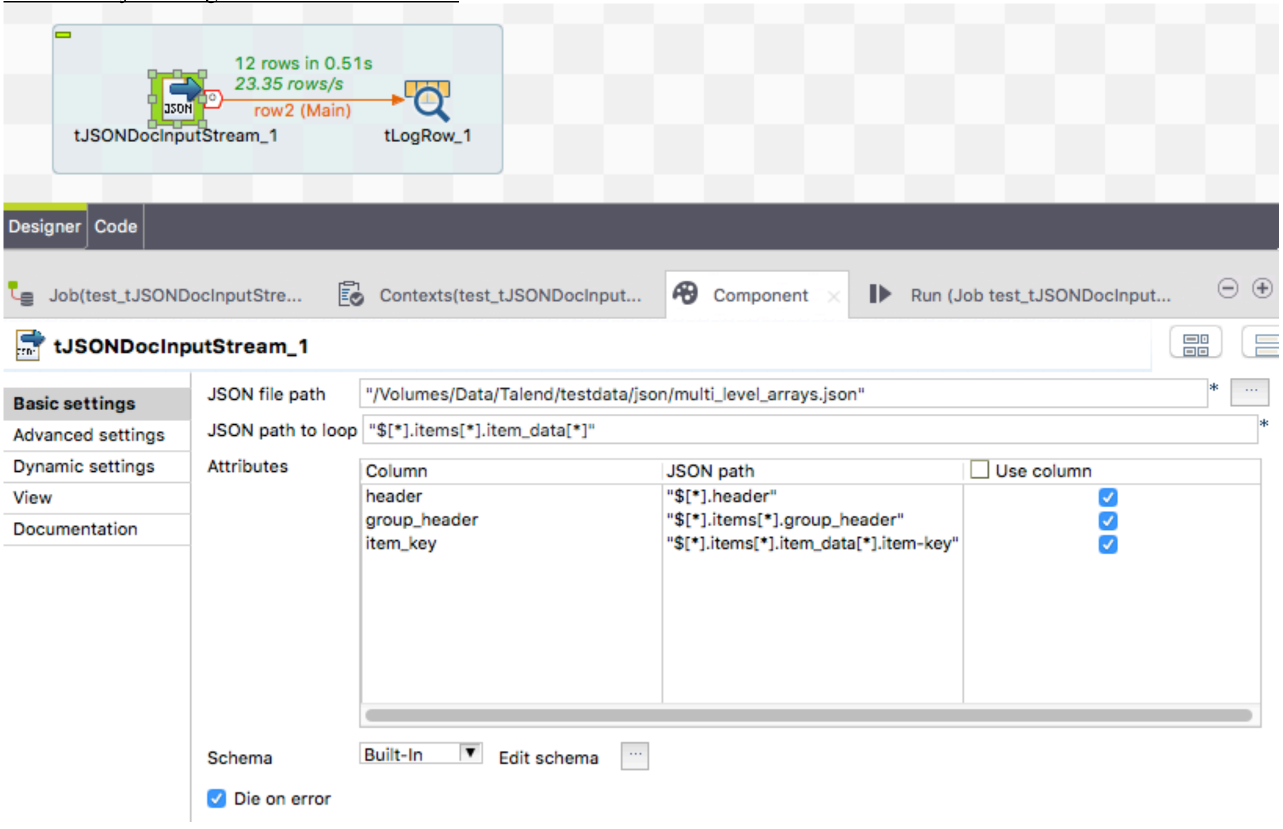
This is the json input:

```
[
  {
    "header": "global header1",
    "items": [
      {
        "group_header": "group_header11",
        "item_data": [
           {"item-key": 111, "item-value" : {"a1" : "b1"}},
           {"item-key": 112},
           {"item-key": 113, "item-value" : {"a3" : "b3"}}
        ]
      },
      {
        "group_header": "group_header12",
        "item_data": [
          {"item-key": 121},
          {"item-key": 122},
          {"item-key": 123}
        ]
      }
    ]
  },
  {
    "header": "global header2",
    "items": [
      {
        "group_header": "group_header21",
        "item_data": [
          {"item-key": 211},
          {"item-key": 212, "item-value" : {"a4" : "b4"}},
          {"item-key": 213}
        ]
      },
      {
        "group_header": "group_header22",
        "item_data": [
          {"item-key": 221},
          {"item-key": 222},
          {"item-key": 223, "item-value" : {"a5" : "b5"}}
        ]
      }
    ]
  }
]
```

Expected Output:

```
.--------------+--------------+--------.
|              tLogRow_1               |
|=-------------+--------------+-------=|
|header        |group_header  |item_key|
|=-------------+--------------+-------=|
|global header1|group_header11|111     |
|global header1|group_header11|112     |
|global header1|group_header11|113     |
|global header1|group_header12|121     |
|global header1|group_header12|122     |
|global header1|group_header12|123     |
|global header2|group_header21|211     |
|global header2|group_header21|212     |
|global header2|group_header21|213     |
|global header2|group_header22|221     |
|global header2|group_header22|222     |
|global header2|group_header22|223     |
'--------------+--------------+--------'
```

This is the job design to achive the results:



tJSONDocInputStream_1

Basic settings | Advanced settings | Dynamic settings | View | Documentation

JSON file path: "/Volumes/Data/Talend/testdata/json/multi_level_arrays.json"

JSON path to loop: "$[*].items[*].item_data[*]"

Attributes

Column	JSON path	Use column
header	"$[*].header"	✓
group_header	"$[*].items[*].group_header"	✓
item_key	"$[*].items[*].item_data[*].item-key"	✓

Schema: Built-In   Edit schema

✓ Die on error