 **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 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 cam perform any transformation.

**Talend-Integration**

You find these components 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 modes 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"

+ " \"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"

+ " \"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"

+ " \"example\_array\" : [ 10, 20 ]\n"

+ " } ]\n"

+ " }\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.

**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"

},

"empty\_value" : "something \n 2 lines",

"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"

}

} ]

}, {

"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 | Last error message. Unfortunately this is not the error message from the actually running job. This message is build from the tRunTask component. The current TAC web service does not provide this message. |
| CURRENT\_NODE | This is the root JsonNode in this case. |

**Scenario 1:**

Simply running a task:

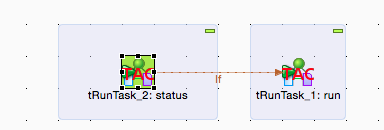
This scenario shows the way to implement a trigger, which starts a task in the TAC for every found file. There are a lot of other scenarios possible.

One of the most used scenarios is to trigger a task from another Job scheduler because of a company policy about scheduling. In large companies there are typically dedicated schedulers and with this component you can write a job with it self can started as simple script from such kind of schedulers.

**Scenario 2: Watchdog job**

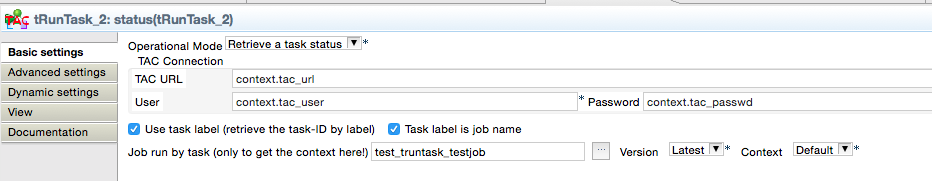
In this scenario a job check the status of a task and if the task has been failed the task will be run again.

In a real world scenario such job will be run also as task in the TAC and would be triggered several times within a hour.



This is the simple job design.

Here the basic settings for the tRunTask\_2. The enhanced label can be achieved by set as label in the View setting: \_\_UNIQUE\_NAME\_\_: \_\_MODE\_\_



The if condition is: ((Boolean)globalMap.get("tRunTask\_2\_HAS\_ERRORS"))

This is a return value and can be put here by drag and drop.

If the task has errors, the task should be run again:

Here the basic settings of the tRunTask\_1:

