

Table of contents

- [Table of contents](#)
- [JSON-Map](#)
 - [Description](#)
 - [Compatibility](#)
 - [Getting Started](#)
 - [Output Connections](#)
 - [Flow Element](#)
 - [General Properties](#)
 - [Working Mode Depending Properties](#)
 - [Multi-line](#)
 - [External File](#)
 - [JSONata](#)
 - [Extended functions](#)
 - [Examples](#)
 - [Multi-line](#)
 - [Multi-line with variable](#)
 - [Multi-line with function](#)
 - [Bindings](#)
 - [Lookup Tables](#)
 - [Extended function uuid](#)
 - [Error handling](#)
 - [Private data](#)

JSON-Map

Description

With JSON-Map, you can map/convert JSON content using a JSONata expression.

JSONata is a very powerful query language that allows you to define functions, variables and lookup tables. This makes it really easy to transform data according to your wishes.

Compatibility

Switch Fall 2022 and higher.

Getting Started

Use one of our sample flows and drop a sample file into the flow.

Output Connections

This app requires one incoming connection - more incoming connections are allowed. The app supports traffic light outgoing connections of the following types:

- Log success: carries the mapped json
- Data error: carries the incoming job if the operation fails at the first attempt.
- Data success: carries the incoming job after the operation succeeds. If there are no data success connections the output is simply suppressed (with logging a warning).

Flow Element

General Properties

Property	Value	Description
<i>DataSource</i>	Enum [Job Dataset]	Defines the source of data. Only a valid JSON is supported.
<i>DatasetName</i>	String	Name of the JSON dataset that should be mapped.
<i>WorkingMode</i>	Enum [Multi-line External File]	Defines if the jsonata expression is defined as multi-line field or as external file
<i>Bindings</i>	String	<p>Define a JSON object where each property can be accessed by \$<key> in the JSONata expression.</p> <p>A binding is any value that's only available during runtime.</p> <p>E.g.</p> <pre>{ "variables": { "key1": "value1", "key2": "value2" } }</pre>
<i>LookupTables</i>	String	<p>Defines <key>=<value> pairs, where the value is the path to json file.</p> <p>The json file will be read and the content of the json is accessible by using \$lookupTables.<key> in the jsonata expression.</p> <p>E.g.:</p> <p>formats=<pathToFormats JSON></p>

Property	Value	Description
<i>LookupDatasets</i>	String	<p>Defines names of JSON datasets where the content is accessible by using \$lookupTables.<datasetName> in the jsonata expression.</p> <p>Each dataset should be entered in a separate line. Make sure the dataset name is not equal to any other lookup table key.</p>
<i>ResultDatasetName</i>	String	Name of the resulting dataset. If a dataset with the same name already exists it will be overwritten.

Working Mode Depending Properties

Multi-line

Property	Value	Description
<i>MappingJsonataMultiline</i>	String	Define a valid JSONata expression that should be used for mapping.

External File

Property	Value	Description
<i>MappingJsonataFile</i>	String	Path to a file that contains a valid JSONata expression.

JSONata

The documentation for JSONata is well written and there is also an exerciser which is highly recommended to be used for testing expressions before using them in Switch. The exerciser gives you instant feedback!

Please note that the extended functions are only available in the Enfocus Switch environment and not in the exerciser!

Documentation: <https://docs.jsonata.org/overview.html>

Exerciser / Tester: <https://try.jsonata.org/>

Extended functions

As powerful as JSONata is, it cannot cover all functions. Therefore we have added some functions to JSONata which may be helpful in an Enfocus Switch environment.

NPM Path module: All functions of the NPM module are available as JSONata function. For example use `$path.delimiter` to get the delimiter of the operating system.

Check out the following link for more details <https://nodejs.org/docs/latest/api/path.html>

NPM UUID module: The following functions of the NPM module are available as JSONata function:

- `NIL`
- `parse`
- `stringify`
- `v1`
- `v3`
- `v4`
- `v5`
- `validate`
- `version`

For example use `$uuid.v1()` to call the `v1` function of the module.

Check out the following link for more details <https://www.npmjs.com/package/uuid>

Examples

Multi-line

Input JSON

```

{
  "ID": 1,
  "FirstName": "Fred",
  "Surname": "Smith",
  "Age": 28,
  "Phone": [
    {
      "type": "home",
      "number": "0203 544 1234"
    },
    {
      "type": "office",
      "number": "01962 001234"
    },
    {
      "type": "office",
      "number": "01962 001235"
    },
    {
      "type": "mobile",
      "number": "077 7700 1234"
    }
  ]
}

```

JSONata expression

This expression joins the value of the properties `FirstName` and `Surname` and searches for the phone number of `type` mobile.

```

{
  "name": FirstName & " " & Surname,
  "mobile": Phone[type = "mobile"].number
}

```

Result

```

{
  "name": "Fred Smith",
  "mobile": "077 7700 1234"
}

```

Multi-line with variable

Input JSON

```
{
  "ID": 1,
  "FirstName": "Fred",
  "Surname": "Smith",
  "Age": 28,
  "Phone": [
    {
      "type": "home",
      "number": "0203 544 1234"
    },
    {
      "type": "office",
      "number": "01962 001234"
    },
    {
      "type": "office",
      "number": "01962 001235"
    },
    {
      "type": "mobile",
      "number": "077 7700 1234"
    }
  ]
}
```

JSONata expression

The following expression defines a variable `fullName` which is then used in the resulting object.

```
(
  $fullName := FirstName & " " & Surname;

  {
    "name": $fullName,
    "mobile": Phone[type = "mobile"].number
  }
)
```

Result

```
{
  "name": "Fred Smith",
  "mobile": "077 7700 1234"
}
```

Multi-line with function

Input JSON

```
{
  "ID": 1,
  "FirstName": "Fred",
  "Surname": "Smith",
  "Age": 28,
  "Phone": [
    {
      "type": "home",
      "number": "0203 544 1234"
    },
    {
      "type": "office",
      "number": "01962 001234"
    },
    {
      "type": "office",
      "number": "01962 001235"
    },
    {
      "type": "mobile",
      "number": "077 7700 1234"
    }
  ]
}
```

JSONata expression

The following expression uses a separate function to get a phone number from the type.


```
(
  $getPhoneNumberFromType := function($type) {(
    $$Phone[type = $type].number
  )});

  {
    "name": FirstName & " " & Surname,
    "mobile": $getPhoneNumberFromType('mobile')
  }
)
```

Result

```
{
  "name": "Fred Smith",
  "mobile": "077 7700 1234"
}
```

Bindings

If you need to pass a variable to the JSONata expression, bindings can be used. A binding can be accessed by using \$ in the expression.

While this may not make much sense when using the multiline mode, it is very helpful if the expression is defined in an external file.

Input JSON

```

{
  "ID": 1,
  "FirstName": "Fred",
  "Surname": "Smith",
  "Age": 28,
  "Phone": [
    {
      "type": "home",
      "number": "0203 544 1234"
    },
    {
      "type": "office",
      "number": "01962 001234"
    },
    {
      "type": "office",
      "number": "01962 001235"
    },
    {
      "type": "mobile",
      "number": "077 7700 1234"
    }
  ]
}

```

JSONata expression

This expression joins the value of the properties `FirstName` and `Surname` and searches for the phone number of `type` mobile.

```

{
  "name": FirstName & " " & Surname,
  "mobile": Phone[type = "mobile"].number,
  "jobName": $variables.jobName
}

```

Bindings

```

{
  "variables": {
    "jobName": "[Job.Name]"
  }
}

```

Result

```
{
  "name": "Fred Smith",
  "mobile": "077 7700 1234",
  "jobName": "contacts.json"
}
```

Lookup Tables

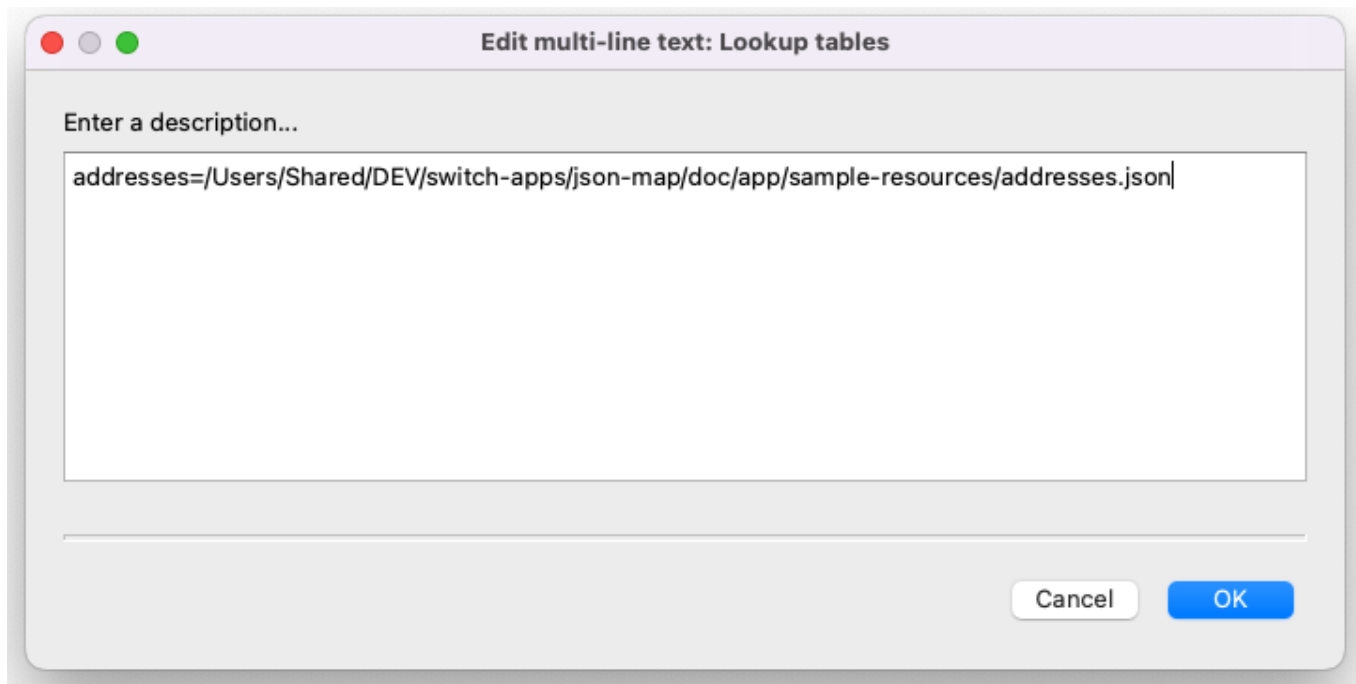
A lookup table can be accessed by using \$lookupTables. in the expression.

Input JSON

```
{
  "ID": 1,
  "FirstName": "Fred",
  "Surname": "Smith",
  "Age": 28,
  "Phone": [
    {
      "type": "home",
      "number": "0203 544 1234"
    },
    {
      "type": "office",
      "number": "01962 001234"
    },
    {
      "type": "office",
      "number": "01962 001235"
    },
    {
      "type": "mobile",
      "number": "077 7700 1234"
    }
  ]
}
```

Lookup JSON

How a lookup table is defined in the app:



Content of lookup table:

```
[
  {
    "customerRefId": 1,
    "street": "2 Long Road",
    "city": "Winchester",
    "postcode": "S022 5PU"
  },
  {
    "customerRefId": 2,
    "street": "56 Letsby Avenue",
    "city": "Winchester",
    "postcode": "S022 4WD"
  },
  {
    "customerRefId": 3,
    "street": "1 Preddy Gate",
    "city": "Southampton",
    "postcode": "S014 0MG"
  }
]
```

JSONata expression

JSONata searches for an element in the lookup table where 'customerRefId' is equal to the 'ID' of the input JSON.

```
{
  "name": FirstName & " " & Surname,
  "mobile": Phone[type = "mobile"].number,
  "address": $lookupTables.addresses[customerRefId=$$.ID]
}
```

Result

```
{
  "name": "Fred Smith",
  "mobile": "077 7700 1234",
  "address": {
    "customerRefId": 1,
    "street": "2 Long Road",
    "city": "Winchester",
    "postcode": "SO22 5PU"
  }
}
```

Extended function uuid

Input JSON

```

{
  "ID": 1,
  "FirstName": "Fred",
  "Surname": "Smith",
  "Age": 28,
  "Phone": [
    {
      "type": "home",
      "number": "0203 544 1234"
    },
    {
      "type": "office",
      "number": "01962 001234"
    },
    {
      "type": "office",
      "number": "01962 001235"
    },
    {
      "type": "mobile",
      "number": "077 7700 1234"
    }
  ]
}

```

JSONata expression

This expression uses the `v1` function of the `uuid` module, which generates a UUID.

```

{
  "id": $uuid.v1(),
  "name": FirstName & " " & Surname,
  "mobile": Phone[type = "mobile"].number
}

```

Result

```

{
  "id": "8ce885c0-3df7-11ef-a6f6-c17f56ffa379",
  "name": "Fred Smith",
  "mobile": "077 7700 1234"
}

```

Error handling

This app uses two types of errors:

- job data: if an handled error occurs (e.g. wrong file format), the error message is logged in the switch messages.
- job fail: if any other error occurs, job will fail and gets sent to the problem jobs folder. The thrown error gets logged as error and can be looked up in the switch messages.

Private data

The following private data tags will be set if an error occurs:

Tag	Value Type	Description
lastErrorElement	String	the name of the flow element
lastErrorId	jsonCreateError	
lastErrorCode	Number	an error code that defines the type of error that occurred
lastErrorMessage	String	detailed error message

Error Codes:

```
enum ERROR_CODES {
  generalError = 0,
  fileHandlingError = 1,
  fileFormatError = 2,
  conversionError = 3,
  invalidParameterValue = 4,
  parsingError = 5,
}
```