Article

← FME Desktop (/S/Topic/0TO4Q000000QL9uWAG/Fme-...

Writing JSON with the JSONTemplater

Aug 4, 2022 • Knowledge

Product Type
FME Desktop

FME Version
2022.0

Tutorial: Tutorial: Getting Started with JSON (/s/article/tutorial-getting-started-with-json) | **Previous:** Writing JSON (/s/article/json-writing-overview) |
Next: Transforming JSON using the JSONExtractor, JSONFlattener, and JSONFragmenter (/s/article/json-transformations)

Introduction

In the previous article, Writing JSON (/s/article/json-writing-overview), we looked at the different ways to write JSON using FME. In this article, we'll look in detail about how to write nested JSON using the JSONTemplater and the Text File writer. This workflow will allow us to write the nested data mentioned in the final step of the previous article.

This use-case is a bit more advanced but is necessary if you want to write nested JSON data. The basic pattern is to read in data (in this case, JSON with no geometry, but it could be any data in FME), use a JSONTemplater to create FME attributes containing JSON with nested structures, and then write the JSON attributes out to a text file. We are using a Text File writer instead of a JSON writer because in FME, any time you want to write out the contents of an attribute directly, you use the Text File writer.

The JSONTemplater uses a template approach to writing nested JSON (very similar to the XMLTemplater (https://docs.safe.com/fme/html/FME_Desktop_Documentation/FME_Transformers/Transformers/xmltemplater.htm)). A template represents the structure of the data, with functions such as fme:get-attribute functions, fme:get-json-attributes or fme:process-features being used within the template to build the JSON structure from FME features. Using sub-templates lets one create a document that has a root with multiple child elements.

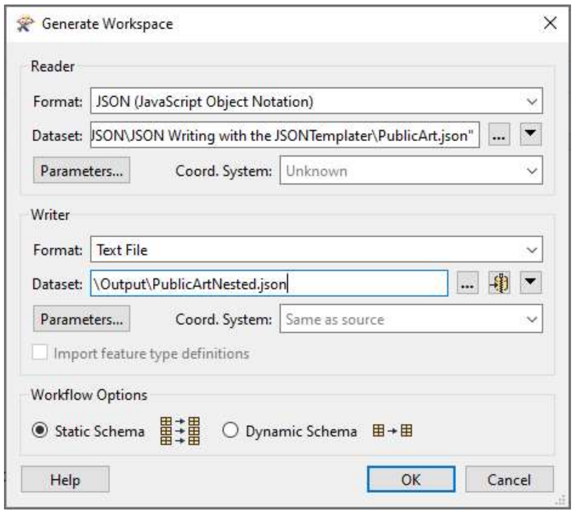
Step-by-step Instructions

1. Start FME Workbench and Generate Workspace

Start FME Workbench and click Generate Workspace. Fill in the Generate Workspace dialog as follows:

- Reader

  - **Format**: JSON (JavaScript Object Notation)
  - **Dataset**: PublicArt.json (download and use local path)
- Writer
  - **Format**: Text File
  - **Dataset**: ...\Output\PublicArtNested.json (where ever you wish)
    You may have to switch the file type to All Files (*) for the .json extension to work.



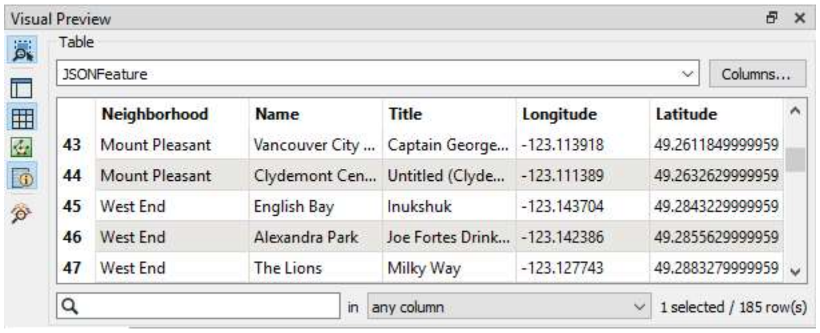
Reader
Format: JSON (JavaScript Object Notation)
Dataset: JSON\JSON Writing with the JSONTemplater\PublicArt.json"
Coord. System: Unknown

Writer
Format: Text File
Dataset: \Output\PublicArtNested.json
Coord. System: Same as source
Import feature type definitions [ ]

Workflow Options
[✓ Static Schema] Dynamic Schema

2. Inspect the Source Data in Visual Preview

Run the workspace and inspect the source dataset



	Neighborhood	Name	Title	Longitude	Latitude
43	Mount Pleasant	Vancouver City ...	Captain George...	-123.113918	49.2611849999959
44	Mount Pleasant	Clydemont Cen...	Untitled (Clyde...	-123.111389	49.2632629999959
45	West End	English Bay	Inukshuk	-123.143704	49.2843229999959
46	West End	Alexandra Park	Joe Fortes Drink...	-123.142386	49.2855629999959
47	West End	The Lions	Milky Way	-123.127743	49.2883279999959

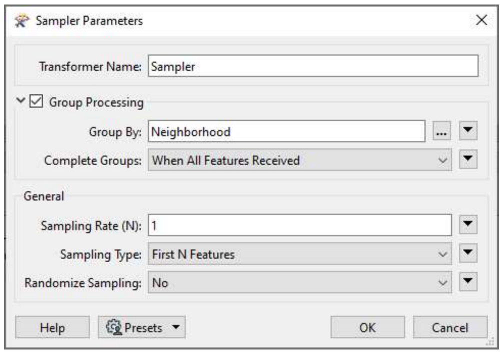
You can see each of the **185 features** is an art installation in Vancouver. Although the data includes longitude and latitude values, we will not be creating geometry in this example. The goal is to transform this JSON from a large array with Neighborhood as an attribute to a nested structure of key-value pairs where each art installation is nested underneath its respective neighborhood, like so:

```
{
    "Downtown": [{
            "Name": "Harbour Centre Parkade",
            "Title": "The Belonging Action",
            "Longitude": -123.110097741722,
            "Latitude": 49.2837806793832
        },
        {
            "Name": "Chinese Cultural Centre",
            "Title": "China Gate",
            "Longitude": -123.103282272368,
            "Latitude": 49.2797561341325
        },
        ...
    ],
    "Strathcona": [{
            "Name": "National  Works Yard",
            "Title": "Roller",
            "Longitude": -123.092675,
            "Latitude": 49.2736209999959
        },
        {
            "Name": "Jim Green Residence",
            "Title": "Entranceway",
            "Longitude": -123.095131,
            "Latitude": 49.2842699999959
        },
        ...
    ],
    ...
}
```

3. Add a Sampler

To build our nested JSON, we will use a JSONTemplater with two sub-templates: one to create a separate array for each neighborhood, and another to contain all the art installations in that neighborhood. To get a separate array for each neighborhood we need to provide the JSONTemplater with six features, one from each neighborhood. To do this, we'll use a Sampler transformer. Add a Sampler and connect it to the reader feature type, then set the following parameters:

- **Group Processing**: Enable
    - **Group By:** Neighborhood
- **Sampling Rate (N)**:1
- **Sampling Type**: First N Features



Sampler Parameters

Transformer Name: Sampler

Group Processing

Group By: Neighborhood

Complete Groups: When All Features Received

General

Sampling Rate (N): 1

Sampling Type: First N Features

Randomize Sampling: No

With Feature Caching enabled, click Run To This on your Sampler. You should have six features coming out of the Sampler:Sampled port, one for each neighborhood. These will be provided to one of the JSONTemplater sub-templates.

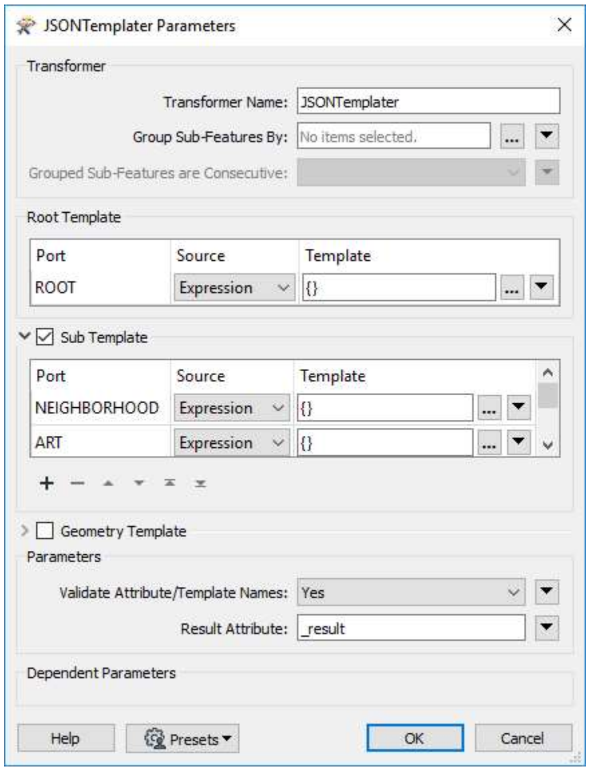
4. Add a JSONTemplater

Now that we have the neighborhood features, add a JSONTemplater after the Sampler, but do not connect it; we need to create input ports first.

Open the JSONTemplater. First, let's add our sub-templates. Sub-templates are used to turn FME features into children of the ROOT template, or even other sub-templates.

To add a sub-template, check the Sub Template box and then click the + button in the Sub Template table. In the Port field, rename this sub-template NEIGHBORHOOD. Add another sub-template and call it ART.

You'll notice each Template field is red, meaning we must supply a value before we can click OK. Because we want to connect our new sub-template ports to features, start by just typing empty curly braces {} into each Template field. Your dialog should look like this:



JSONTemplater Parameters

Transformer

Transformer Name: JSONTemplater

Group Sub-Features By: No items selected.

Grouped Sub-Features are Consecutive:

Root Template

Port	Source	Template
ROOT	Expression	{}

Sub Template

Port	Source	Template
NEIGHBORHOOD	Expression	{}
ART	Expression	{}

Geometry Template

Parameters

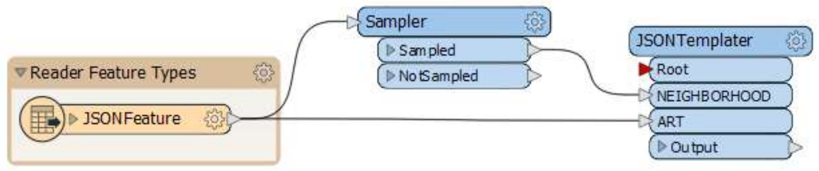
Validate Attribute/Template Names: Yes

Result Attribute: _result

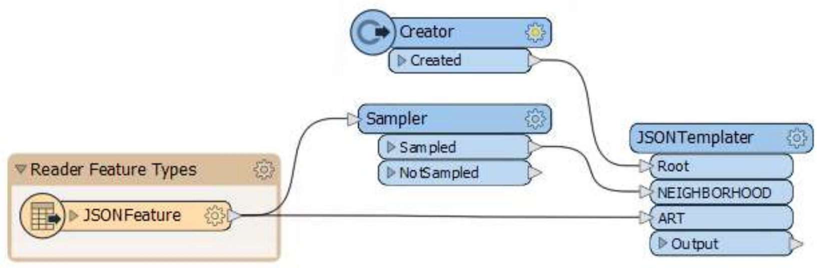
Dependent Parameters

5. Connect Features to the JSONTemplater Input Ports

You'll see that each template (ROOT, NEIGHBORHOOD, and ART) creates an input port on the JSONTemplater. We are supplying six features to ROOT, but currently none to the other ports. Connect the Sampled port on the Sampler to the NEIGHBORHOOD input port. Then since we want ART to contain all of the art installations, connect the reader feature type to the ART input port. This will provide all 185 features for use in our sub-template.



Now we need to connect features to the Root template. Because we want a single JSON document to be output from the JSONTemplater, the easiest thing to do is to use a Creator, which will provide a single feature to the Root template. Add a Creator to the canvas and connect it to the Root input port of the JSONTemplater.



6. Build the ROOT Template

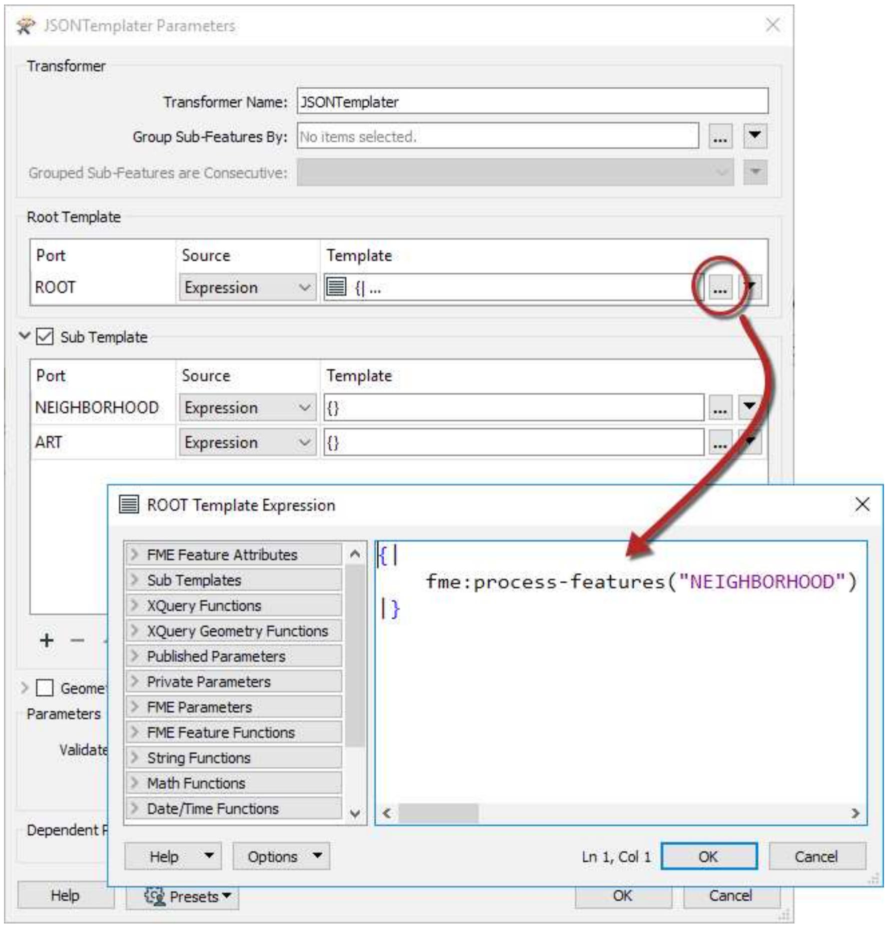
Now that we have features connected to both our templates, open the parameters for the JSONTemplater again.

First, let's set up the ROOT template. This template sets the top level of the JSON hierarchy, and in this case, will only execute once because it is receiving one feature. Click on the ellipse [...] button to the ROOT Template field to open the ROOT Template Expression dialog. This dialog is similar to the Text Editor and lets us build our Template Expression. Copy and paste the following template, or build it yourself by typing and double-clicking the Sub Templates > SUB on the left to add functions (note the pipes inside the curly braces, {| |}):

```
{|
    fme:process-features("NEIGHBORHOOD")
|}
```

The fme:process-features("NEIGHBORHOOD") function will insert the results of our sub-template NEIGHBORHOOD as items in the array.

For this example, we are using one of the slightly more advanced JSON templating expressions, the pipe |. As specified in the JSONiq documentation (http://www.jsoniq.org/docs/JSONiqExtensionToXQuery/html-single/index.html#idm48114976), the pipe is a dynamic object construction expression. What it means, in this case is, create each result of the sub-template as a separate object. If we didn't add these pipes, the resulting JSON document would not have the required commas between each neighborhood entry.

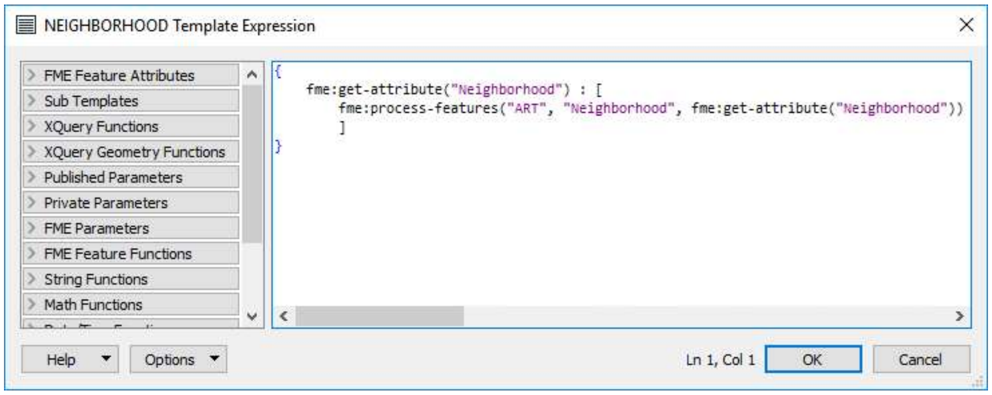


```
{|
    fme:process-features("NEIGHBORHOOD")
|}
```

7. Build the NEIGHBORHOOD template

This sub-template will create the JSON document for each feature, with the name of the neighborhood being substituted for fme:get-attribute("Neighborhood"). The fme:process-features(“ART”) function will insert the results of our sub-template ART as items in the array. Click on the ellipses [...] button next to the Template field for NEIGHBORHOOD. Copy and paste the following template:

```
{
    fme:get-attribute("Neighborhood") : [
        fme:process-features("ART", "Neighborhood", fme:get-attribute("Neighborhood"))
        ]
}
```

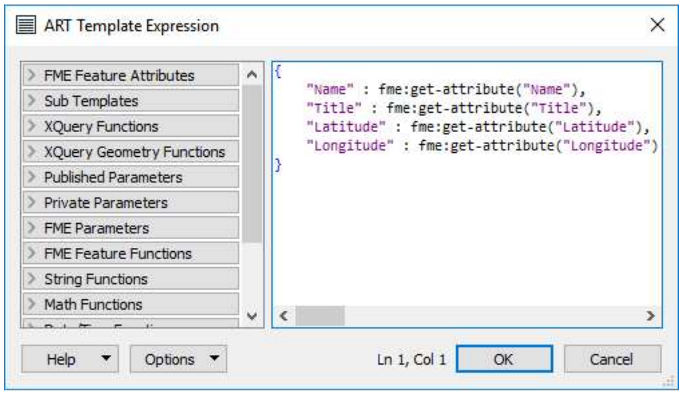


```
{
    fme:get-attribute("Neighborhood") : [
        fme:process-features("ART", "Neighborhood", fme:get-attribute("Neighborhood"))
        ]
}
```

8. Build the ART template

Next, let’s build the ART sub-template. Open the Template Expression for ART**.** Copy and paste the following template:

```
{
    "Name" : fme:get-attribute("Name"),
    "Title" : fme:get-attribute("Title"),
    "Latitude" : fme:get-attribute("Latitude"),
    "Longitude" : fme:get-attribute("Longitude")
}
```

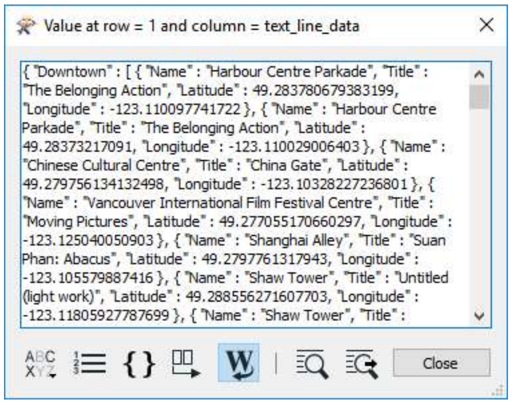


```
{
    "Name" : fme:get-attribute("Name"),
    "Title" : fme:get-attribute("Title"),
    "Latitude" : fme:get-attribute("Latitude"),
    "Longitude" : fme:get-attribute("Longitude")
}
```

This template will build the array of art installation data, with each FME feature being turned into a piece of JSON matching this pattern. Click OK.

9. View the Results

Run the workspace and inspect the Output port of the JSONTemplater. You should see all the neighborhoods combined into one JSON document matching our goal template in the attribute _result. Click the ellipse [...] button next to the cell in the Table View of Visual Preview to view the full value. However, this JSON is quite hard to read as it is not pretty-printed:

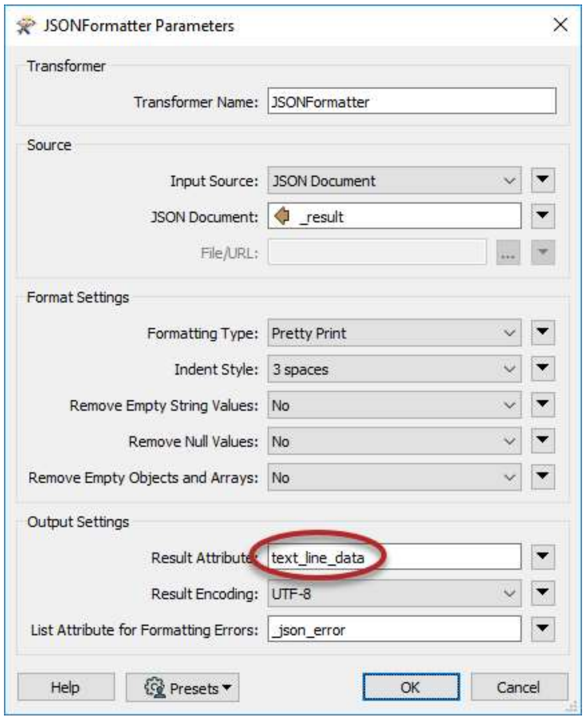


```
{ "Downtown" : [ { "Name" : "Harbour Centre Parkade", "Title" :
"The Belonging Action", "Latitude" : 49.283780679383199,
"Longitude" : -123.110097741722 }, { "Name" : "Harbour Centre
Parkade", "Title" : "The Belonging Action", "Latitude" :
49.28373217091, "Longitude" : -123.110029006403 }, { "Name" :
"Chinese Cultural Centre", "Title" : "China Gate", "Latitude" :
49.279756134132498, "Longitude" : -123.10328227236801 }, {
"Name" : "Vancouver International Film Festival Centre", "Title" :
"Moving Pictures", "Latitude" : 49.277055170660297, "Longitude" :
-123.125040050903 }, { "Name" : "Shanghai Alley", "Title" : "Suan
Phan: Abacus", "Latitude" : 49.2797761317943, "Longitude" :
-123.105579887416 }, { "Name" : "Shaw Tower", "Title" : "Untitled
(light work)", "Latitude" : 49.288556271607703, "Longitude" :
-123.11805927787699 }, { "Name" : "Shaw Tower", "Title" :
```

Let's fix that issue.

10. Add a JSONFormatter

Add a JSONFormatter after your JSONTemplater. This transformer will format our JSON document so it is pretty-printed. Open its parameters and set the JSON Document to _result. Then, set the Output Attribute to text_line_data. This attribute name is reserved for use when writing using the Text File writer. Your dialog should look like this:



JSONFormatter Parameters

Transformer

Transformer Name:	JSONFormatter

Source

Input Source:	JSON Document
JSON Document:	_result
File/URL:	

Format Settings

Formatting Type:	Pretty Print
Indent Style:	3 spaces
Remove Empty String Values:	No
Remove Null Values:	No
Remove Empty Objects and Arrays:	No

Output Settings

Result Attribute:	text_line_data
Result Encoding:	UTF-8
List Attribute for Formatting Errors:	_json_error

11. Run Workspace and Inspect Final Results

Connect the JSONFormatter to the Text File writer feature type. Run your workspace and inspect the final results, either in Visual Preview or your text editor of choice. You should see JSON following the goal structure identified at the beginning of the article, something like this (abbreviated):

```
{
    "Downtown": [{
            "Name": "Harbour Centre Parkade",
            "Title": "The Belonging Action",
            "Longitude": -123.110097741722,
            "Latitude": 49.2837806793832
        },
        {
            "Name": "Chinese Cultural Centre",
            "Title": "China Gate",
            "Longitude": -123.103282272368,
            "Latitude": 49.2797561341325
        },
        ...
    ],
    "Strathcona": [{
            "Name": "National  Works Yard",
            "Title": "Roller",
            "Longitude": -123.092675,
            "Latitude": 49.2736209999959
        },
        {
            "Name": "Jim Green Residence",
            "Title": "Entranceway",
            "Longitude": -123.095131,
            "Latitude": 49.2842699999959
        },
        ...
    ],
    ...
}
```

Congratulations! You learned how to create custom nested JSON from FME features using the JSONTemplater. Combining sub-templates with the functions available in the Template Expression dialog allow for complex custom JSON results.

Continue to the next article: Transforming JSON using the JSONExtractor, JSONFlattener, and JSONFragmenter (/s/article/json-transformations)

Data Attribution

The data used here originates from open data made available by the City of Vancouver (https://opendata.vancouver.ca/pages/home/), British Columbia. It contains information licensed under the Open Government License - Vancouver.

First Published Date
7/29/2020, 12:18 AM

Last Published Date
8/4/2022, 1:59 AM

Transformation (/s/topic/0TO4Q000000... FME Desktop (/s/topic/0TO4Q000000...

Sort by:
Latest Posts

LizAtSafe (/s/profile/0050c00000CeGV0AAN) (Employee) published a new version of this Knowledge.
August 4, 2022 at 1:59 AM (/s/feed/0D54Q00009ggWD6SAM)

1 view

Like Comment

Log In to Comment

kailinatsafe (/s/profile/0054Q00000EwiZmQAJ) (Employee) published a new version of this Knowledge.
July 14, 2022 at 8:14 PM (/s/feed/0D54Q00009e2sfwSAA)

1 view

Like Comment

Log In to Comment

jnotter (/s/profile/0054Q00000EwwD1QAJ)
January 31, 2020 at 9:33 PM (/s/feed/0D54Q000080hmvbSAA)

@samatsafe, @deanatsafe This has been a very insightful article for a project I am working in which I render JSON data to embed in a MapBox GL HTML template (not leaflet). One key problem I am seeing however is that adding the pipes on the root template triggers an error message, but without the pipes I am not getting the commas. Not sure how to troubleshoot this.

Additionally, my template generates polygons, but the coordinates output is wrapped in quotes... how do I get rid of the quotes?

1 comment 53 views

Like Comment

samatsafe (/s/profile/0050c00000DIg3tAAD) (Employee)
3 years ago

Hi @jnotter (https://knowledge.safe.com/users/9626/jnotter.html), glad to hear you found the article useful. A few thoughts for your issue:

- Is there a reason you are writing to JSON instead of GeoJSON? I imagine MapBox could ingest the GeoJSON just as easily and it might simplify the writing step in FME, simply using the GeoJSON writer. If it's a matter of needing a nested data structure, you can write an attribute that contains a string of JSON and give that attribute a type of JSON on the GeoJSON writer feature type. Just a thought.
- Assuming you do need to write to JSON, I'd recommend you check out the JSONiq documentation. I find working with the object constructors can be a bit tricky, but if you look at these examples (http://www.jsoniq.org/docs/JSONiqExtensionToXQuery/html-single/index.html#idm48114976) it might help you find the right syntax to get your desired

Expand Post

Like

Log In to Comment

amolparande (/s/profile/0054Q00000EwvSCQAZ)
March 28, 2019 at 11:48 AM (/s/feed/0D54Q000080hmp2SAA)

@petrahammoser i have question related " JSONTemplater sub-template configuration "
"type" : "LineString",
"coordinates" :[] how should we parse LineString in to "coordinates" ? can you please help me ?

32 views

Like Comment

Log In to Comment

imoliviasmith (/s/profile/0054Q00000EwwatQAB)
May 18, 2018 at 2:24 PM (/s/feed/0D54Q000080hmKmSAI)

Awesome article, I use JSON Formatter (https://jsonformatter.org/) for validating and formatting JSON data.

1 comment 25 views

Like Comment

helmoet (/s/profile/0054Q00000Ewv6AQAR) (Partner)
2 years ago

Indeed inspiring article. However, what if you don't know the schema and want for the Art object just all the attributes in the JSON array?

Like

Log In to Comment

Follow

Files (3) (/s/relatedlist/ka14Q000001DWyXQAW/AttachedContentDocuments)

jsontemplater-2022
Jun 24, 2022 • 83KB • fmw

publicartnested
Jul 28, 2020 • 33KB • json

publicart
Jul 28, 2020 • 31KB • json

View All (/s/relatedlist/ka14Q000001DWyXQAW/AttachedContentDocuments)

Related Articles

Writing JSON (/s/article/json-writing-overview)

Tutorial: Getting Started with JSON (/s/article/tutorial-getting-started-with-json)

Transforming JSON using the JSONExtractor, JSONFlattener, and JSONFragmenter (/s/article/json-transformations)

How to Convert Parquet to JSON (/s/article/How-to-Convert-Parquet-to-JSON)

Extracting Location from JSON (/s/article/converting-from-json-to-a-spatial-format-gis)

SAFE SOFTWARE®
(https://safe.com)

Getting Started (../s/topic/0TO4Q000000QKioWAG/welcome) Ideas (../s/bridea/acideasULT brIdea c/00B4Q000000fwHEVLA1) Feedback (https://forms.gle/LkoFWpziDYaWQkL78)

Forums (../s/forums/) Groups (../s/group/CollaborationGroup/00Ba000000A0BxJEAV) Knowledge Base (../s/knowledge-base/) Support (../s/support/)

Register / Log In (/s/login/)