Transforming JSON data with the help of Azure Data Factory - Part 4 - Passing Arrays around

Rayis Imayev, 2020-06-09 (first published: 2020-05-24)

(2020-May-24) It has never been my plan to write a series of articles about how I can work with JSON files in Azure Data Factory (ADF). While working with one particular ADF component I then had discovered other possible options to use richness and less constrained JSON file format, which in a nutshell is just a text file with one or more (***"key" : "value"***) pair elements.

Sometimes you could have an array of those pair elements or an array of other arrays; now you see where I'm going with the name of my current blog post 🙂

Previous blog posts on using JSON files in Azure Data Factory:
Part 1: Transforming JSON to CSV with the help of Azure Data Factory - Mapping Data Flows
Part 2: Transforming JSON to CSV with the help of Azure Data Factory - Wrangling Data Flows
Part 3: Transforming JSON to CSV with the help of Azure Data Factory - Control Flows

There are several ways how you can explore the JSON way of doing things in the Azure Data Factory. The first two that come right to my mind are:
(1) ADF activities' output - they are JSON formatted
(2) Reading JSON files - the task itself produces JSON formatted results too

It's not a new thing to know that we can reference nested elements of ADF activities' output since it's represented in JSON format or pass the JSON file content to other tasks/components that can process this format.

But what if you need to pass a complete output of your ADF activity task further down your pipeline. Or you need to pass a JSON array elements to another ADF activity or sub-pipeline as the parameter value. Let's explore what other options available in Azure Data Factory for this very interesting use case.

(1) ADF activities' output

Let's say I have a "Get Metadata" task in my ADF pipeline that reads the content of a particular folder in my storage account.

The main portion of the output for this activity may look this way:

I already know that I can get access to the list of files/folders using this ADF expression:
@activity('List of files and folders').output.childitems

Or getting to the first file by referencing a very first element [0] of this JSON array:
@activity('List of files and folders').output.childitems[0]

However, if I want to store the complete output of this activity into a separate Array variable with preserving the option of referencing all nested elements, then this can only be done by wrapping the output into array by using the function with the same name:
@array(activity('List of files and folders').output)

Otherwise you will get a data mismatch error message. This will become very helpful if you want to pass the JSON output into the next activity of your ADF pipeline to process. It works and it's fun! 🙂

(2) Reading JSON files

Let's get more creative and read a real JSON file where you have more flexibility and control over its content.

Let's say I want to read the following JSON file of various baking lists of ingredients: https://opensource.adobe.com/Spry/samples/data_region/JSONDataSetSample.html

```
[
 {
"id": "0001",
"type": "donut",
"name": "Cake",
"ppu": 0.55,
"batters":
   {
"batter":
     [
      { "id": "1001", "type": "Regular" },
      { "id": "1002", "type": "Chocolate" },
      { "id": "1003", "type": "Blueberry" },
      { "id": "1004", "type": "Devil's Food" }
     ]
   },
"topping":
   [
    { "id": "5001", "type": "None" },
    { "id": "5002", "type": "Glazed" },
    { "id": "5005", "type": "Sugar" },
    { "id": "5007", "type": "Powdered Sugar" },
    { "id": "5006", "type": "Chocolate with Sprinkles" },
    { "id": "5003", "type": "Chocolate" },
    { "id": "5004", "type": "Maple" }
   ]
 },
 {
"id": "0002",
"type": "donut",
"name": "Raised",
"ppu": 0.55,
"batters":
   {
```

```
"batter":
     [
      { "id": "1001", "type": "Regular" }
     ]
   },
"topping":
   [
    { "id": "5001", "type": "None" },
    { "id": "5002", "type": "Glazed" },
    { "id": "5005", "type": "Sugar" },
    { "id": "5003", "type": "Chocolate" },
    { "id": "5004", "type": "Maple" }
   ]
 },
 {
"id": "0003",
"type": "donut",
"name": "Old Fashioned",
"ppu": 0.55,
"batters":
   {
"batter":
     [
      { "id": "1001", "type": "Regular" },
      { "id": "1002", "type": "Chocolate" }
     ]
   },
"topping":
   [
    { "id": "5001", "type": "None" },
    { "id": "5002", "type": "Glazed" },
    { "id": "5003", "type": "Chocolate" },
    { "id": "5004", "type": "Maple" }
   ]
 }
]
```

The output of my ADF Lookup activity that references this file

will correctly show me 3 elements of this JSON array:

Where a list of items for my "For Each" loop activity can be simply set with the following expression:
@activity('Reading JSON file').output.value

Here is the interesting part, let's say I want to execute another ADF pipeline within my "For Each" loop activity and pass one baking recipe (or list of ingredients) as a parameter. Azure Data Factory is flexible enough and I can accomplish this with the following expression:
@array(item())

My sub-pipeline accepts this array parameter value and does further JSON data elements referencing jobs:

I can save incoming parameter value into (*var_baking_payload* variable) with this expression:
@pipeline().parameters.param_baking_payload

List of toppings (*var_toppings* variable) can be set with this expression:
@variables('var_baking_payload')[0].topping

List of batters (*var_batters* variable) can be set with this expression:
@variables('var_baking_payload')[0].batters.batter

The thing that I got excited the most while working with JSON file outputs in Data Factory pipelines was that I could still pass JSON arrays between different tasks/activities or pass those arrays to another sub-pipeline as a parameter, and nested

elements referencing still worked!

If I'm the last one to learn this, I'm still excited 🙂

Happy Data Adventures!

Original post (opens in new tab)