



# Azure Data Factory Multiple File Load Example - Part 2

By: [Koen Verbeeck \(/sqlserverauthor/114/koen-verbeeck/\)](#) | Updated: 2020-02-03 | [Comments \(12\)](#) | Related: [More \(/sql-server-developer-resources/\)](#) > [Azure Data Factory \(/sql-server-tip-category/245/azure-data-factory/\)](#)

## Problem

In this two-part tip, we are created a metadata-driven pipeline which will copy multiple flat files from Azure blob storage to an Azure SQL Database. The flat files can have different delimiters and are stored in different folders and there are multiple destination tables as well.

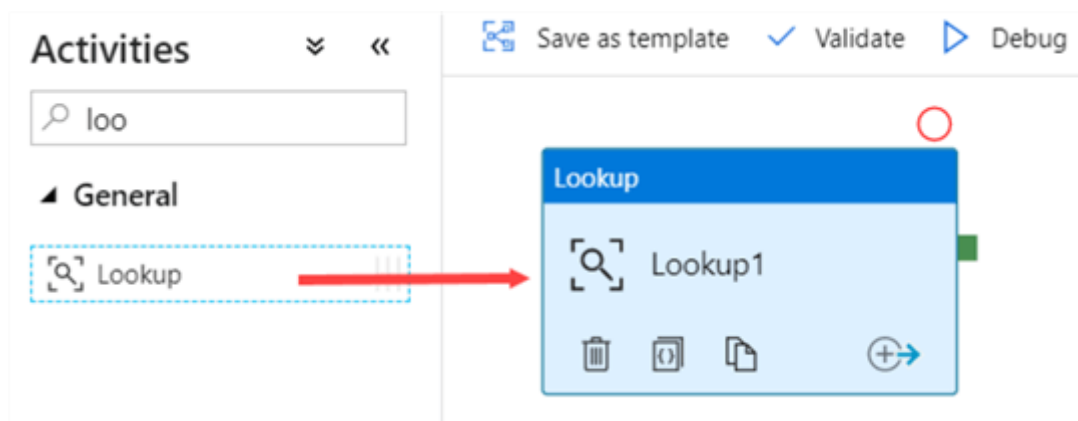
## Solution

In [part 1 of this tip \(/sqlservertip/6281/how-to-load-multiple-files-in-parallel-in-azure-data-factory--part-1/\)](#), we created the metadata table in SQL Server and we also created parameterized datasets in Azure Data Factory. In this part, we will combine both to create a metadata-driven pipeline using the ForEach activity.

If you want to follow along, make sure you have [read part 1 for the first step \(/sqlservertip/6281/how-to-load-multiple-files-in-parallel-in-azure-data-factory--part-1/\)](#).

### Step 2 – The Pipeline

With the datasets ready, we can now start on the pipeline. The first action is retrieving the metadata. In a new pipeline, drag the **Lookup activity** to the canvas.



With the following query, we can retrieve the metadata from SQL Server:

```

SELECT
    b.[ObjectName]
    ,FolderName = b.[ObjectValue]
    ,SQLTable    = s.[ObjectValue]
    ,Delimiter   = d.[ObjectValue]
FROM [dbo].[Metadata_ADF] b
JOIN [dbo].[Metadata_ADF] s ON b.[ObjectName] = s.[ObjectName]
JOIN [dbo].[Metadata_ADF] d ON b.[ObjectName] = d.[ObjectName]
WHERE b.[SourceType] = 'BlobContainer'
      AND s.[SourceType] = 'SQLTable'
      AND d.[SourceType] = 'Delimiter';

```

The advantage of datasets is you can use them dynamically with parameters, but you can also use it for any query connecting to the same database. This means we can use the dataset we created before as a placeholder for this metadata query (as long as the metadata and the destination tables are in the same database of course).

[General](#)
[Settings](#)
[User properties](#)

Source dataset \*

sql\_movies\_dynamic

Open
New
Preview data

Dataset properties ⓘ

NAME	VALUE	TYPE
TableName	<div>_notSet</div>	string

Use query

☐ Table
☒ Query
☐ Stored Procedure

Query \*

SELECT
 b.[ObjectName]
 ,FolderName = b.[ObjectValue]

Query timeout (minutes)

120 ⓘ

First row only

☐

All we have to do is change the *Use query* property to **Query** and fill in the query below. Since the query is returning more than one row, don't forget to deselect the *First row only* checkbox. We did define a parameter on the table name though, so we need to specify some value, otherwise an error is returned. This can be solved by specifying a dummy value, such as `_notSet` in this example.

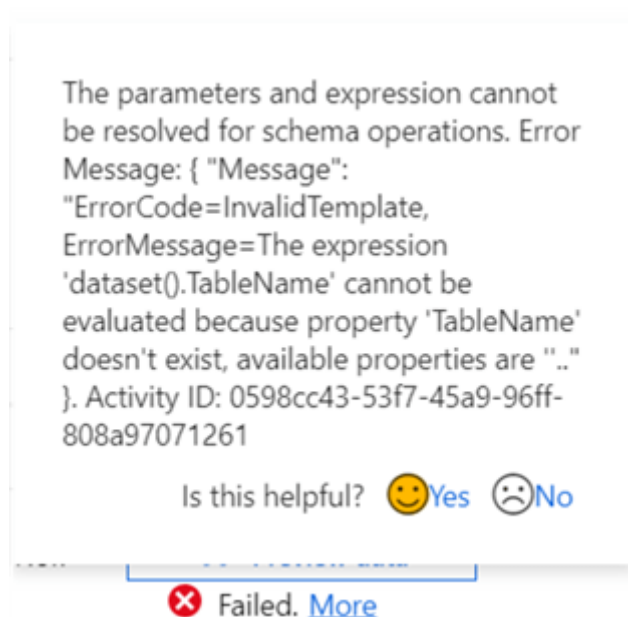
We can preview the data:

## Preview data

Linked service: SQLTest

ObjectName	FolderName	SQLTable	Delimiter
semicolondata	semicolon	topmovies_semicolon	;
commadata	comma	topmovies_comma	,

If the dummy table name was not specified, the following error would have been returned by the preview:



Next, we add a **ForEach** iterator to the pipeline and connect it with the Lookup activity.



On the settings pane of the ForEach activity, click on *Add dynamic content*.


General **Settings**<sup>1</sup> Activities(0) User properties

Sequential ☐

Batch count

Items \* 

This property should be parameterized.

 [Add dynamic content \[Alt+P\]](#)

Here we're going to select the output of the Lookup activity.

### Add dynamic content

@activity('Retrieve Metadata').output.value

[Clear contents](#)

Use [expressions, functions](#) or refer to [system variables](#).

Pipeline trigger type  
Type of the trigger that invoked the pipeline (Manual, Scheduler)

▲ Functions

- ⌵ [Expand all](#)
- Collection Functions
- Conversion Functions
- Date Functions
- Logical Functions
- Math Functions
- String Functions

▲ Activity outputs

Retrieve Metadata  
Retrieve Metadata activity output

At the end of the expression, add `.value` so the full expression becomes:

```
@activity('Retrieve Metadata').output.value
```

The settings tab should now look like this:

General **Settings** Activities(0) User properties

Sequential ☐

Batch count

Items

Go to the *Activities* tab and click on **Add activity**.

General Settings **Activities(0)** User properties

**Add activity**

Activity type	Activity name
No records found	

This will take you to a new pipeline canvas, where we add the **Copy Data** activity.

**Activities**

Move & transform

Save as template Validate Debug Add trigger

DynamicPipeline > Loop over Metadata





Copy Blob to SQL

You can go back to the original pipeline by selecting the *DynamicPipeline* link at the top of the canvas. Go to the Source tab of the Copy Data activity and select the *csv\_movie\_dynamic* dataset. You have to specify the parameter values for the FolderName and the DelimiterSymbol parameters. This can be done using the following expression:

@{item().ObjectValue}

Here *ObjectValue* is a metadata column from the Lookup activity. You need to replace it with the actual column name you need. The tab for the source then becomes:

General **Source** Sink<sup>1</sup> Mapping Settings User properties

Source dataset \*  csv\_movies\_dynamic  Open  New  Preview data

Dataset properties ⓘ

NAME	VALUE	TYPE
FolderName	@{item().FolderName}	string
DelimiterSymbol	@{item().Delimiter}	string




Recursively ☒ ⓘ

Wildcard folder path

Wildcard file name

A wildcard for the file name was also specified, to make sure only csv files are processed. For the sink, we need to specify the `sql_movies_dynamic` dataset we created earlier.

General Source **Sink** Mapping Settings User properties

Sink dataset \*  sql\_movies\_dynamic  Open  New

Dataset properties ⓘ

NAME	VALUE	TYPE
TableName	@{item().SQLTable}	string


Stored procedure name   Refresh ☐ Edit ⓘ

Table option ☒ None ☐ Auto create table ⓘ

Pre-copy script  ⓘ

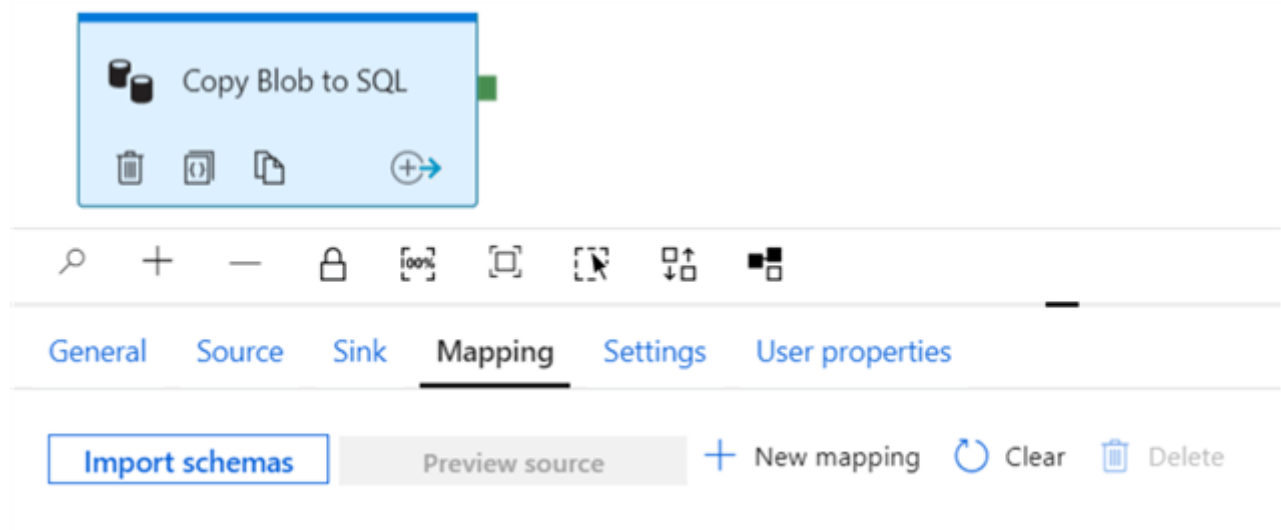
Here, we need to specify the parameter value for the table name, which is done with the following expression:

```
@{item().SQLTable}
```

It's also possible to specify a SQL statement that will be executed at the start of the Copy Data activity, before any data is loaded. In the Pre-copy script field, we can specify an expression that will truncate the destination table with the following expression:

```
TRUNCATE TABLE dbo.{item().SQLTable}
```

Make sure no mapping is specified at all in the Mapping tab. Again, this set-up will only work if the columns in the flat file are the same as in the destination table.



### Step 3 – Debugging The Pipeline

Go back to the original pipeline (containing the Lookup and the ForEach) and hit the debug button at the top. If everything went successful, you'll get an output like this:

Pipeline run ID: **2d1ff384-29e0-4a19-8c19-8b3f68f2881d** [🔍] [🔄] [🚫]

NAME	TYPE	RUN START	DURATION	STATUS	ACTIONS	RUN ID
Copy Blob to SQL	Copy	2019-12-19T09:34:19.084	00:00:05	✔ Succeeded	🔍 🔄 ⚙️	dd1ff391-e6
Copy Blob to SQL	Copy	2019-12-19T09:34:18.984	00:00:05	✔ Succeeded	🔍 🔄 ⚙️	f1cd9ddf-17
Loop over Metadata	ForEach	2019-12-19T09:34:18.203	00:00:08	✔ Succeeded	🔍	9c31aa7e-1c
Retrieve Metadata	Lookup	2019-12-19T09:34:14.343	00:00:04	✔ Succeeded	🔍 🔄	5fe04cee-76

By clicking on the output arrow for the Retrieve Metadata activity, we can view the metadata retrieved from the SQL Server database:

**Output**

```
{
  "count": 2,
  "value": [
    {
      "ObjectName": "semicolondata",
      "FolderName": "semicolon",
      "SQLTable": "topmovies_semicolon",
      "Delimiter": ";"
    },
    {
      "ObjectName": "commadata",
      "FolderName": "comma",
      "SQLTable": "topmovies_comma",
      "Delimiter": ","
    }
  ]
}
```

For the ForEach iterator, we can verify two items were declared in the input:

## Input

```
{
  "ItemsCount": "2"
}
```

For the input of one of the CopyData activities, we can verify the truncate table statement for the sink was constructed correctly:

## Input

```
{
  "source": {
    "type": "DelimitedTextSource",
    "storeSettings": {
      "type": "AzureBlobStorageReadSettings",
      "recursive": true,
      "wildcardFileName": "*.csv",
      "enablePartitionDiscovery": false
    },
    "formatSettings": {
      "type": "DelimitedTextReadSettings"
    }
  },
  "sink": {
    "type": "AzureSqlSink",
    "preCopyScript": "TRUNCATE TABLE dbo.topmovies_semicolon",
    "disableMetricsCollection": false
  },
  "enableStaging": false
}
```

For the output of the CopyData activity (for the files with a semicolon), we can see two files were indeed processed in parallel:

## Output

```
{
  "dataRead": 7347,
  "dataWritten": 12522,
  "filesRead": 2,
  "sourcePeakConnections": 2,
  "sinkPeakConnections": 2,
  "rowsRead": 350,
  "rowsCopied": 350,
  "copyDuration": 4,
  "throughput": 1.794,
  "errors": [],
  "effectiveIntegrationRuntime": "DefaultIntegrationRuntime (West Europe)",
  "usedDataIntegrationUnits": 4,
  "billingReference": "{ \"activityType\": \"DataMovement\", \"billableDuration\": [ { \"Managed\": 0.066",
  "usedParallelCopies": 2,
}
```

While for the other CopyData activity, only one file (with a comma) was processed:



## Output

```
{
  "dataRead": 1082,
  "dataWritten": 1820,
  "filesRead": 1,
  "sourcePeakConnections": 1,
  "sinkPeakConnections": 2,
  "rowsRead": 50,
  "rowsCopied": 50,
  "copyDuration": 3,
  "throughput": 0.352,
  "errors": [],
  "effectiveIntegrationRuntime": "DefaultIntegrationRuntime (West Europe)",
  "usedDataIntegrationUnits": 4,
  "billingReference": "{\"activityType\":\"DataMovement\",\"billableDuration\":{\"Managed\":0.06",
  "usedParallelCopies": 1,
```

When you click on the glasses next to one of the CopyData activities in the debug output, you can see more performance related information:

### Details



Refresh

[Learn more on copy performance details from here.](#) Provide [feedback](#) on performance.

Activity run id: 475bc20f-3186-4421-992f-90dd1383db65



**Azure Blob Storage**  
Region: West Europe

Succeeded



**Azure SQL Database**  
Region: West Europe

Data read: 4.162 KB  
Files read: 2  
Rows read: 200  
Peak connections: 2

Data written: 7.082 KB  
Rows written: 200  
Peak connections: 2  
Throughput: 1.041 KB/s

Copy duration 00:00:04

▲ Azure Blob Storage → Azure SQL Database Queue ⓘ 00:00:01 | Pre copy script ⓘ 00:00:00 | Transfer ⓘ 00:00:02

Start time 12/19/19, 10:45:34 AM  
Duration 00:00:04  
Used DIUs 4  
Used parallel copies 2

As you can see, two files were processed with a total of 200 rows for that one specific CopyData activity.

It's important to notice there are actually two parallel operations in this set-up:

1. The ForEach loop will process each blob folder separately in parallel (semicolon files vs comma files).

2. In an iteration of the ForEach loop, the CopyData activity itself will process all blob files found in one folder also in parallel (2 files found with semicolon data).

## Conclusion

In this tip we saw how we can build a metadata-driven pipeline in Azure Data Factory. Using metadata stored in a key-value pair table in SQL Server, we can use this data to populate parameters of the different datasets. Because of those parameters, we can use a single dataset dynamically to pick up multiple types of files or write to different tables in a database. The ForEach iterator loops over the metadata and executes a CopyData activity dynamically.

## Next Steps

- If you want to follow along, you can download the flat files with the movie titles [here \(/tipImages2/6282\\_SourceFiles.zip\)](#). Make sure you also have [read part 1 of this tip series \(/sqlservertip/6281/how-to-load-multiple-files-in-parallel-in-azure-data-factory--part-1/\)](#).
- For more information about the activities in Azure Data Factory, check out the following tips:
  - [Azure Data Factory ForEach Activity Example \(/sqlservertip/6187/azure-data-factory-foreach-activity-example/\)](#)
  - [Azure Data Factory Get Metadata Example \(/sqlservertip/6246/azure-data-factory-get-metadata-example/\)](#)
  - [Azure Data Factory Lookup Activity Example \(/sqlservertip/6185/azure-data-factory-lookup-activity-example/\)](#)
  - [Azure Data Factory Control Flow Activities Overview \(/sqlservertip/6137/azure-data-factory-control-flow-activities-overview/\)](#)
- You can find more Azure tips in [this overview \(/sql-server-tip-category/87/azure/\)](#).

Last Updated: 2020-02-03

## About the author



[\(/sqlserverauthor/114/koen-verbeeck/\)](#) Koen Verbeeck is a BI professional, specializing in the Microsoft BI stack with a particular love for SSIS.

[View all my tips \(/sqlserverauthor/114/koen-verbeeck/\)](#)

### Related Resources

- [More Database Developer Tips... \(/sql-server-developer-resources/\)](#)