Parameters in Azure Data Factory

In the last mini-series inside the series (🙃), we will go through how to build dynamic pipelines in Azure Data Factory. In this post, we will look at parameters, expressions, and functions. Later, we will look at variables, loops, and lookups. Fun!

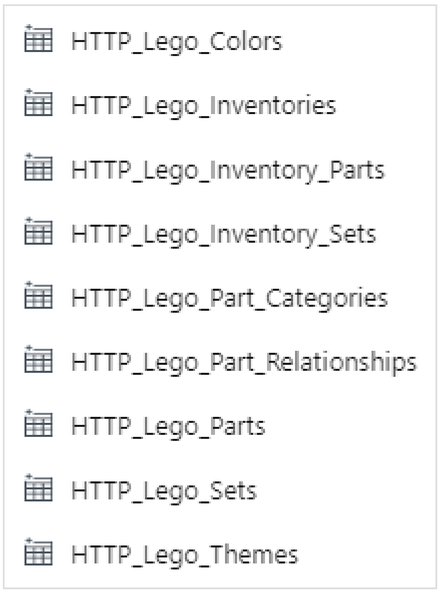
But first, let's take a step back and discuss why we want to build dynamic pipelines at all.

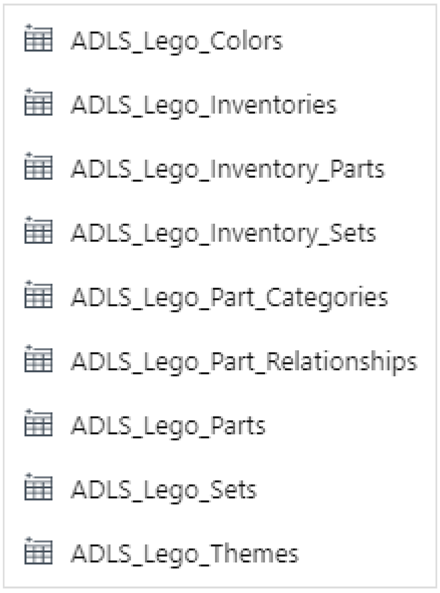
Hardcoded Solutions

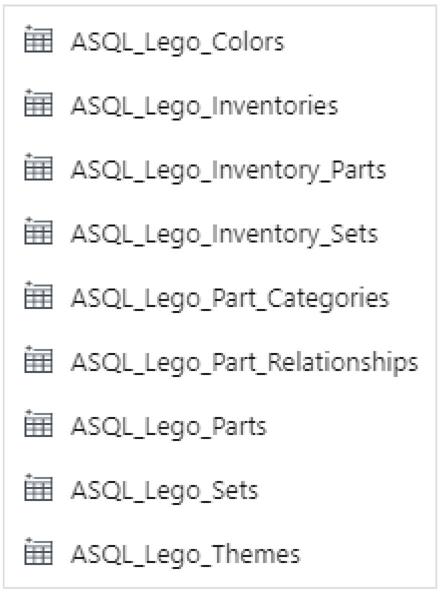
Back in the post about the copy data activity, we looked at our demo datasets. The LEGO data from Rebrickable consists of nine CSV files. So far, we have hardcoded the values for each of these files in our example datasets and pipelines.

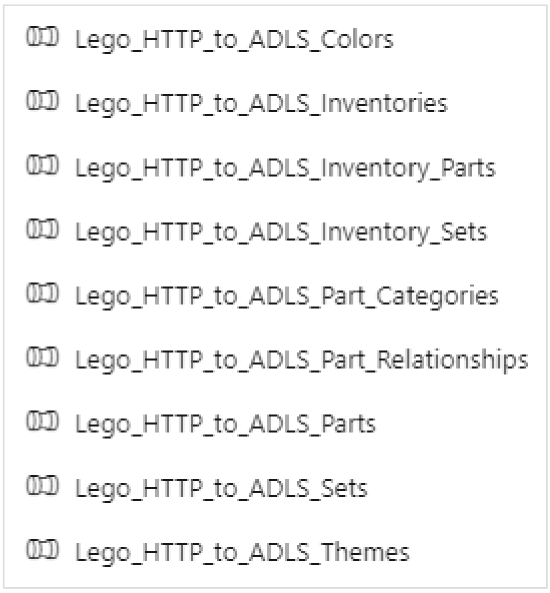
Now imagine that you want to copy all the files from Rebrickable to your Azure Data Lake Storage account. Then copy all the data from your Azure Data Lake Storage into your Azure SQL Database. What will it look like if you have to create *all* the individual datasets and pipelines for these files?

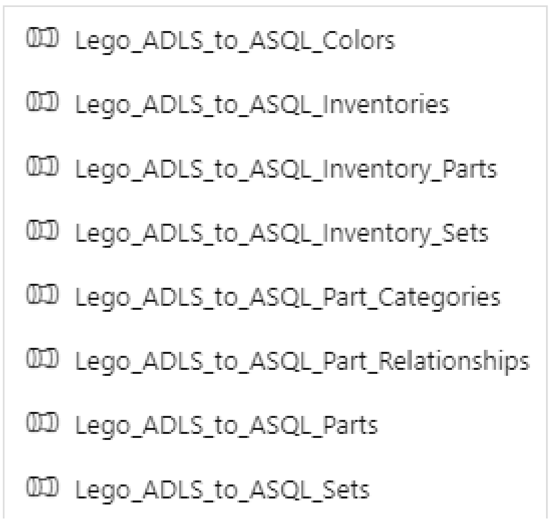
Like this. It will look like this:


HTTP_Lego_Colors
HTTP_Lego_Inventories
HTTP_Lego_Inventory_Parts
HTTP_Lego_Inventory_Sets
HTTP_Lego_Part_Categories
HTTP_Lego_Part_Relationships
HTTP_Lego_Parts
HTTP_Lego_Sets
HTTP_Lego_Themes


ADLS_Lego_Colors
ADLS_Lego_Inventories
ADLS_Lego_Inventory_Parts
ADLS_Lego_Inventory_Sets
ADLS_Lego_Part_Categories
ADLS_Lego_Part_Relationships
ADLS_Lego_Parts
ADLS_Lego_Sets
ADLS_Lego_Themes


ASQL_Lego_Colors
ASQL_Lego_Inventories
ASQL_Lego_Inventory_Parts
ASQL_Lego_Inventory_Sets
ASQL_Lego_Part_Categories
ASQL_Lego_Part_Relationships
ASQL_Lego_Parts
ASQL_Lego_Sets
ASQL_Lego_Themes


Lego_HTTP_to_ADLS_Colors
Lego_HTTP_to_ADLS_Inventories
Lego_HTTP_to_ADLS_Inventory_Parts
Lego_HTTP_to_ADLS_Inventory_Sets
Lego_HTTP_to_ADLS_Part_Categories
Lego_HTTP_to_ADLS_Part_Relationships
Lego_HTTP_to_ADLS_Parts
Lego_HTTP_to_ADLS_Sets
Lego_HTTP_to_ADLS_Themes


Lego_ADLS_to_ASQL_Colors
Lego_ADLS_to_ASQL_Inventories
Lego_ADLS_to_ASQL_Inventory_Parts
Lego_ADLS_to_ASQL_Inventory_Sets
Lego_ADLS_to_ASQL_Part_Categories
Lego_ADLS_to_ASQL_Part_Relationships
Lego_ADLS_to_ASQL_Parts
Lego_ADLS_to_ASQL_Sets

Hooboy! I don't know about you, but I do *not* want to create all of those resources! 😵

(And I mean, I *have* created all of those resources, and then some. I currently have 56 hardcoded datasets and 72 hardcoded pipelines in my demo environment, because I have demos of *everything*. And I don't know about you, but I *never* want to create all of those resources again! 😂)

So! What can we do instead?

Dynamic Solutions

We can build **dynamic solutions**!

Creating hardcoded datasets and pipelines is not a bad thing in itself. It's only when you start creating many *similar* hardcoded resources that things get tedious and time-consuming. Not to mention, the risk of manual errors goes drastically up when you feel like you create the same resource over and over and over again.

(Trust me. When I got to demo dataset #23 in the screenshots above 👆, I had pretty much tuned out and made a bunch of silly mistakes. I went through that so you won't have to! 😅)

And *that's* when you want to build dynamic solutions. When you can **reuse patterns** to **reduce development time** and **lower the risk of errors** :)

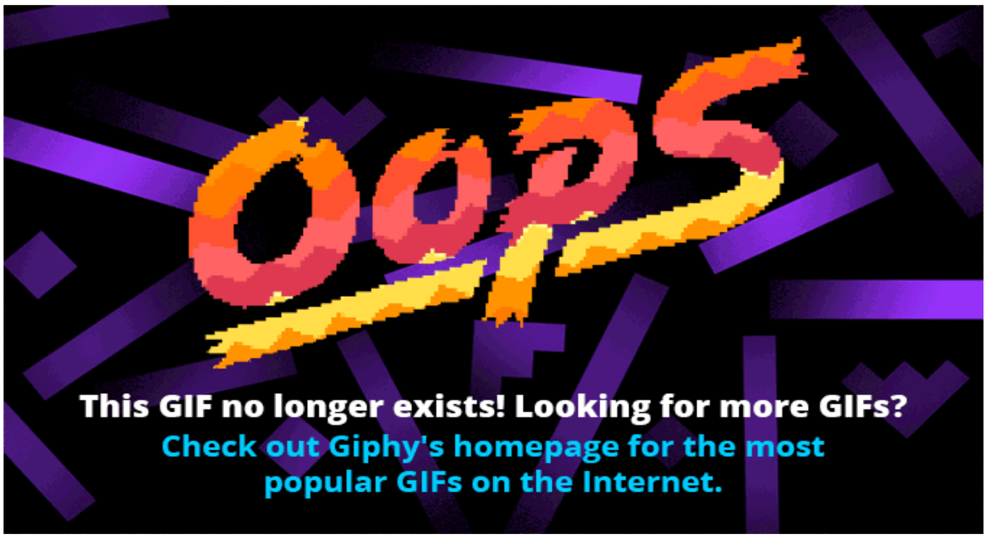
How dynamic should the solution be?

It can be oh-so-tempting to want to build *one solution to rule them all*. (Especially if you love tech and problem-solving, like me. It's *fun* figuring things out!) But be mindful of how much time you spend on the solution itself. If you start spending more time figuring out how to make your solution work for all sources and all edge-cases, or if you start getting lost in your own framework… *stop*.

> Your solution should be dynamic enough that you save time on development and maintenance, but not so dynamic that it becomes difficult to understand.

…don't try to make a solution that is generic enough to solve *everything* :)

Your goal is to **deliver business value**. If you end up looking like this cat, spinning your wheels and working hard (and maybe having lots of fun) but *without getting anywhere*, you are probably over-engineering your solution.


Oops
This GIF no longer exists! Looking for more GIFs?
Check out Giphy's homepage for the most popular GIFs on the Internet.

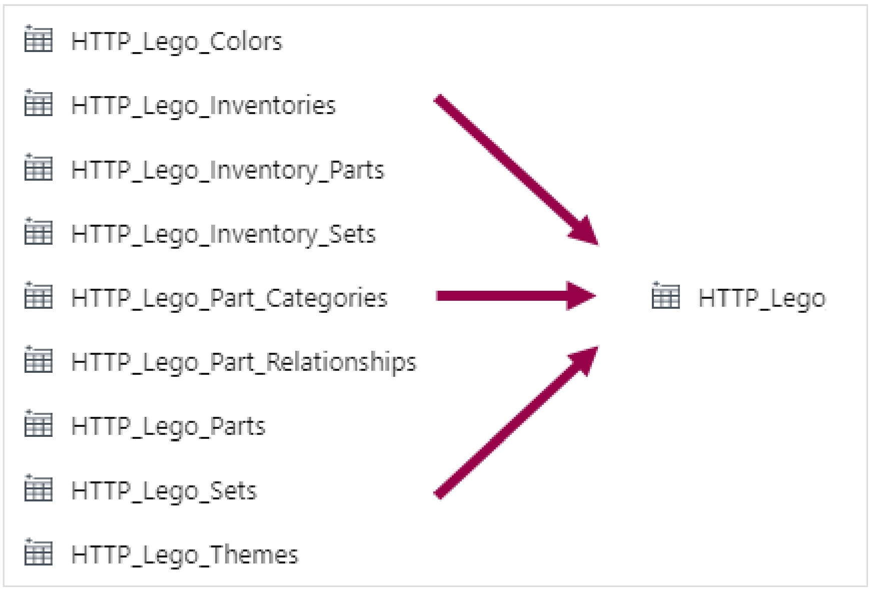
Alright, now that we've got the warnings out the way… Let's start by looking at parameters :)

Parameters

You can use parameters to pass external values into pipelines, datasets, linked services, and data flows. Once the parameter has been passed into the resource, it cannot be changed. By parameterizing resources, you can reuse them with different values each time.

For example, instead of hardcoding the file name from Rebrickable in each dataset, we can parameterize the file name value. Then, we can pass the file name in as a parameter each time we use the dataset.

That means that we can go from *nine* datasets to *one* dataset:



And now we're starting to save some development time, huh? :D

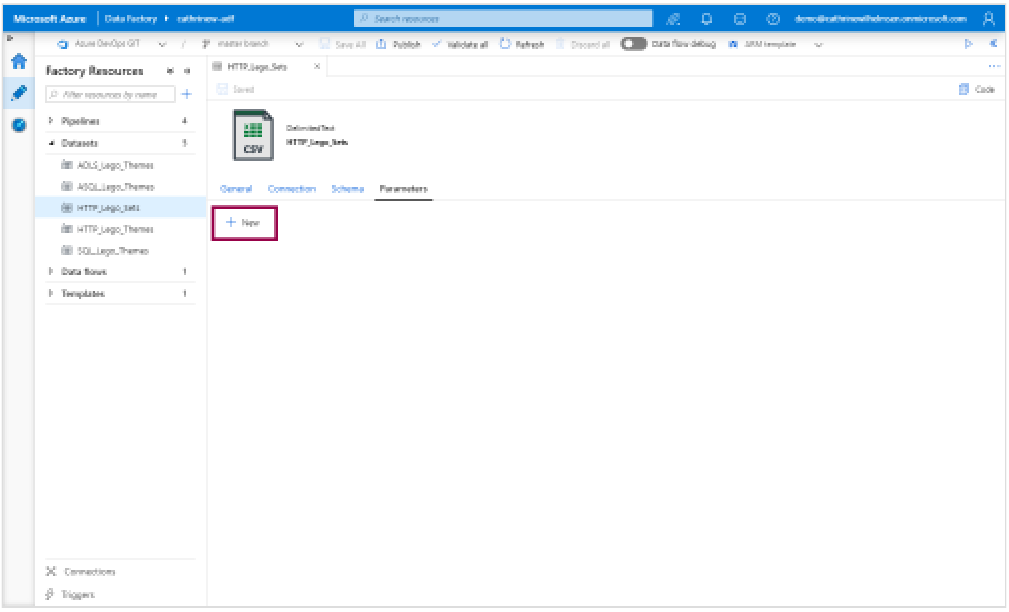
Let's look at how to parameterize our datasets.

Dataset Parameters

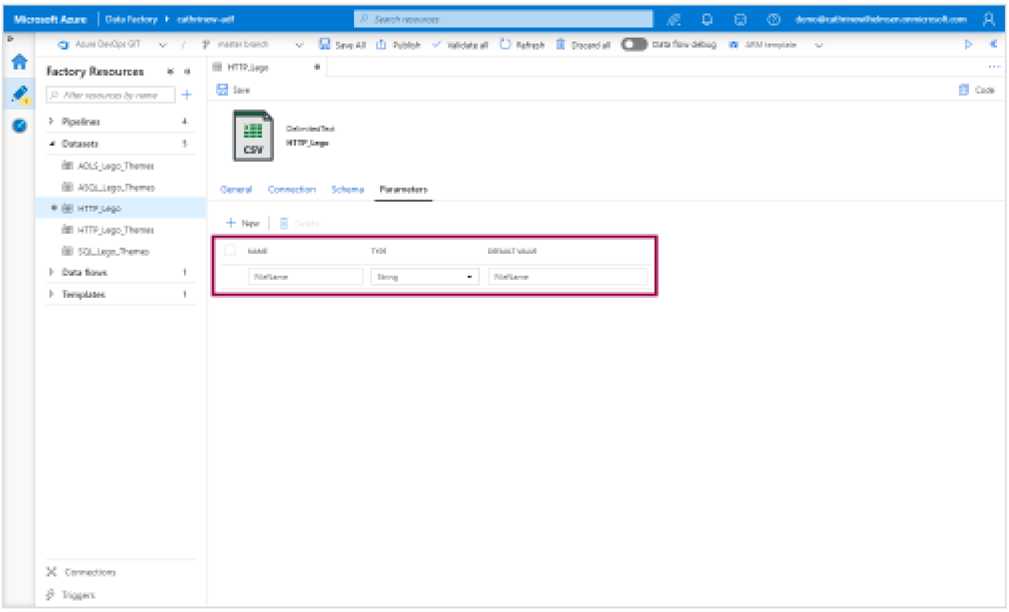
I have previously created two datasets, one for *themes* and one for *sets*. I'm going to change *sets* to be a generic dataset instead.

(*Oof, that was a lot of "sets". I should probably have picked a different example* 😅 *Anyway!*)

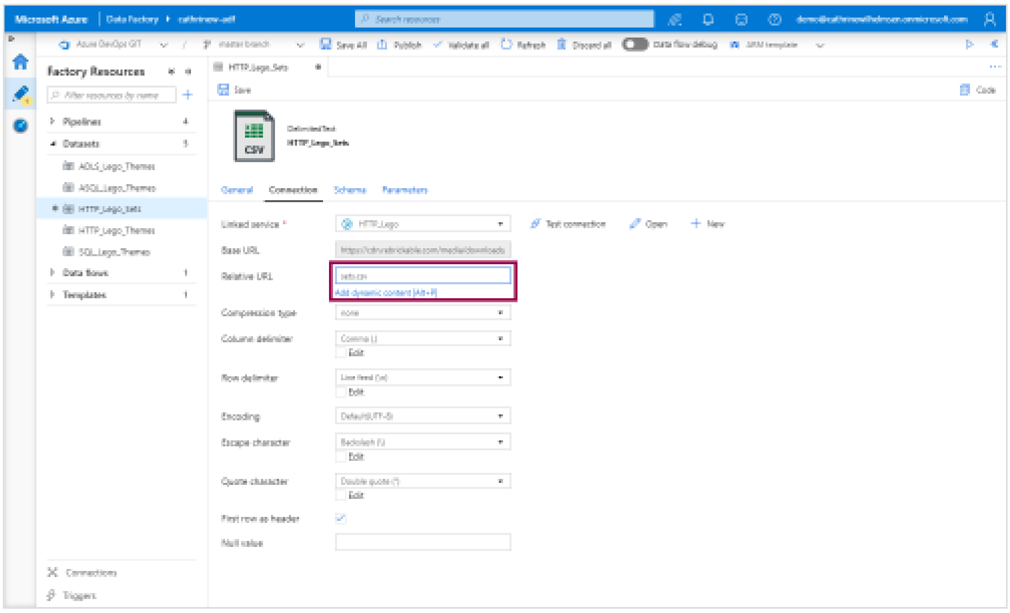
Open the dataset, go to the **parameters** properties, and click **+ new**:



Add a new parameter named **FileName**, of type **String**, with the default value of **FileName**:



Go to the **connection** properties and click inside the **relative URL** field. The **add dynamic content** link will appear under the text box:



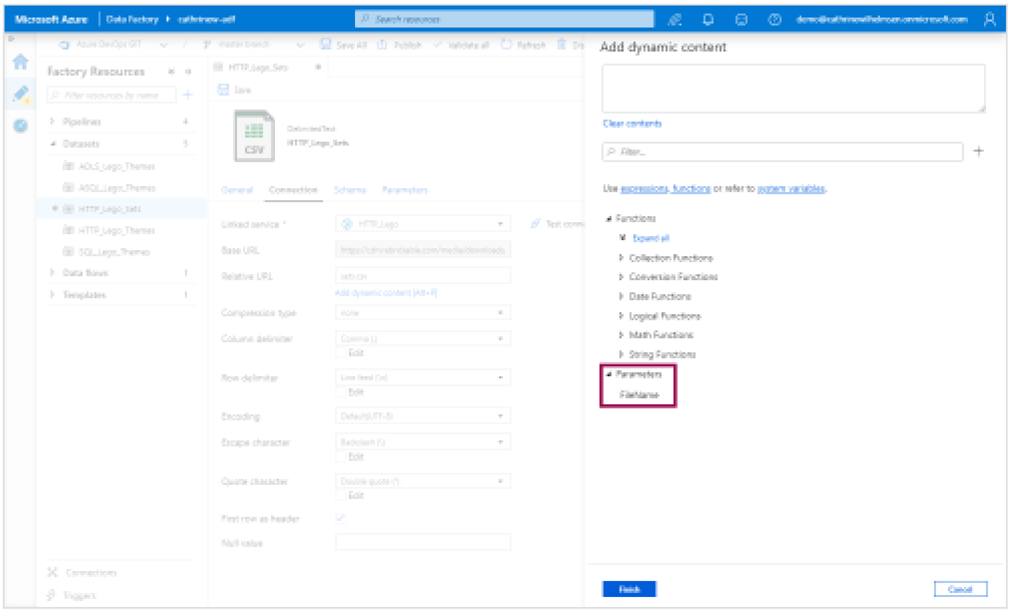
HTTP_Lego_Sets

DelimitedText
HTTP_Lego_Sets

General | Connection | Schema | Parameters

Linked service *: HTTP_Lego
Base URL: https://cdn.rebrickable.com/media/downloads
Relative URL: sets.csv
Add dynamic content [Alt+P]
Compression type: none
Column delimiter: Comma (,)
Row delimiter: Line feed (\n)
Encoding: Default(UTF-8)
Escape character: Backslash (\)
Quote character: Double quote (")
First row as header: ✓
Null value:

When you click the link (or use ALT+P), the **add dynamic content pane** opens. Click the new **FileName** parameter:



Add dynamic content

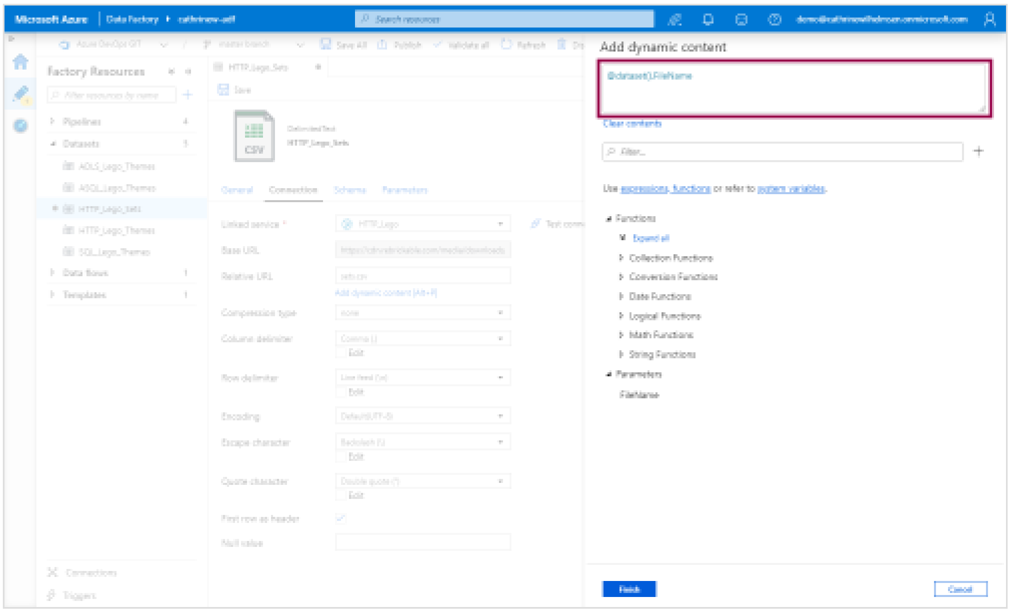
Clear contents

Use expressions, functions or refer to system variables.

- Functions
  - Expand all
  - Collection Functions
  - Conversion Functions
  - Date Functions
  - Logical Functions
  - Math Functions
  - String Functions
- Parameters
  - FileName

Finish | Cancel

The FileName parameter will be added to the dynamic content. Notice the **@dataset().FileName** syntax:



Add dynamic content

@dataset().FileName

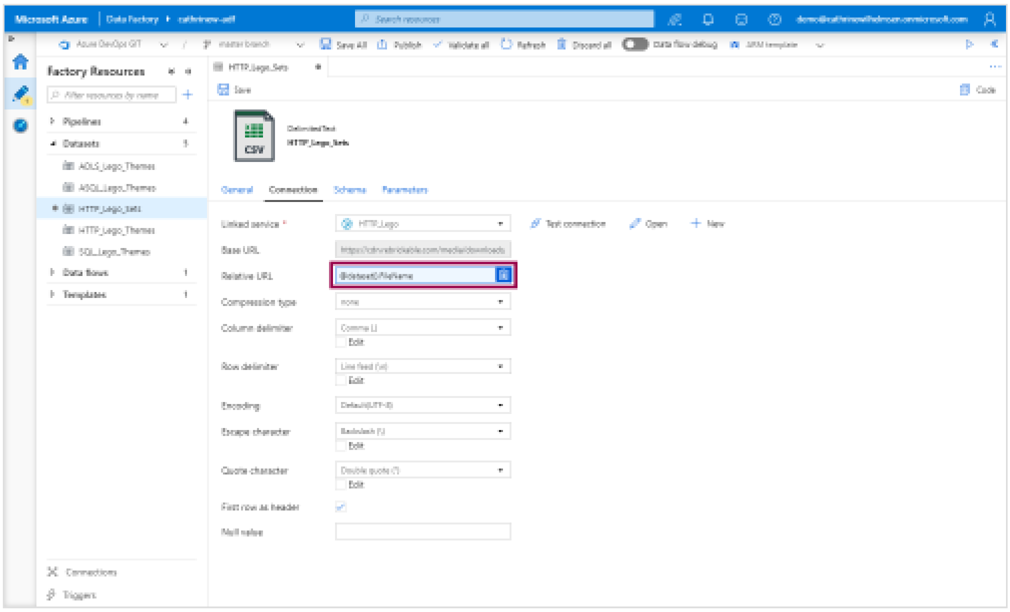
Clear contents

Use expressions, functions or refer to system variables.

- Functions
  - Expand all
  - Collection Functions
  - Conversion Functions
  - Date Functions
  - Logical Functions
  - Math Functions
  - String Functions
- Parameters
  - FileName

Finish Cancel

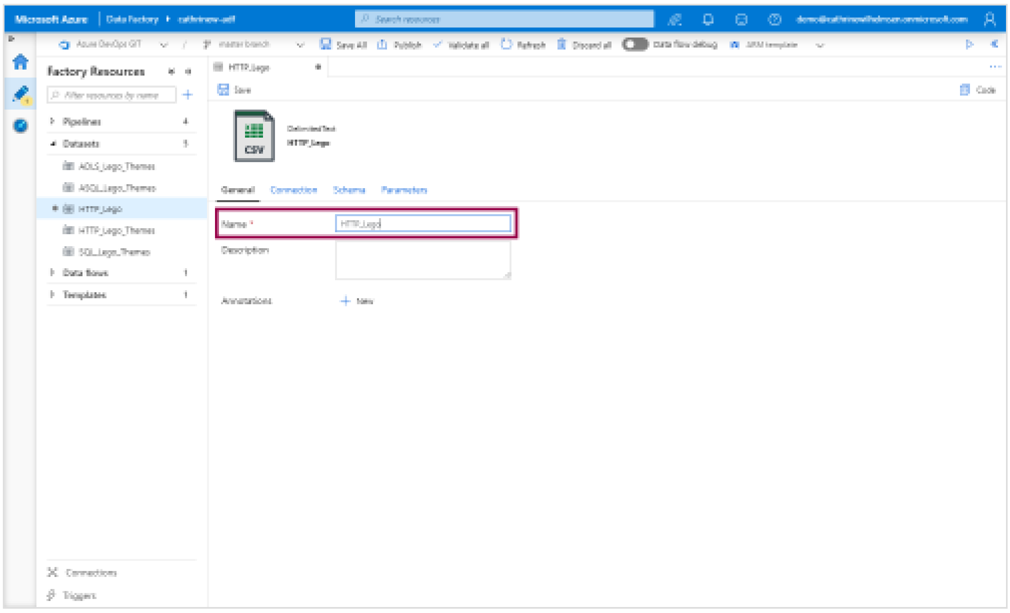
When you click finish, the **relative URL field will use the new parameter**. Notice that the box turns blue, and that a delete icon appears. This shows that the field is using dynamic content. You can click the delete icon to clear the dynamic content:



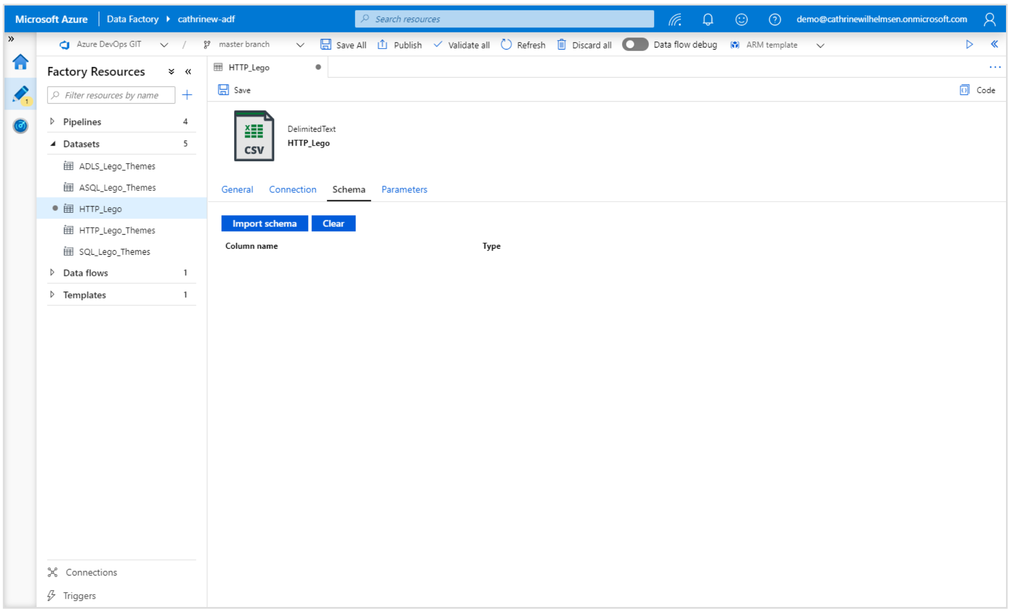
DelimitedText
HTTP_Lego_Sets

Linked service *	HTTP_Lego
Base URL	https://cdn.rebrickable.com/media/downloads
Relative URL	@dataset().FileName
Compression type	none
Column delimiter	Comma (,)
Row delimiter	Line feed (\n)
Encoding	Default(UTF-8)
Escape character	Backslash (\)
Quote character	Double quote (")
First row as header	✓
Null value	

Finally, go to the **general** properties and change the **dataset name** to something more generic:



…and double-check that there is no **schema** defined, since we want to use this dataset for different files and schemas:

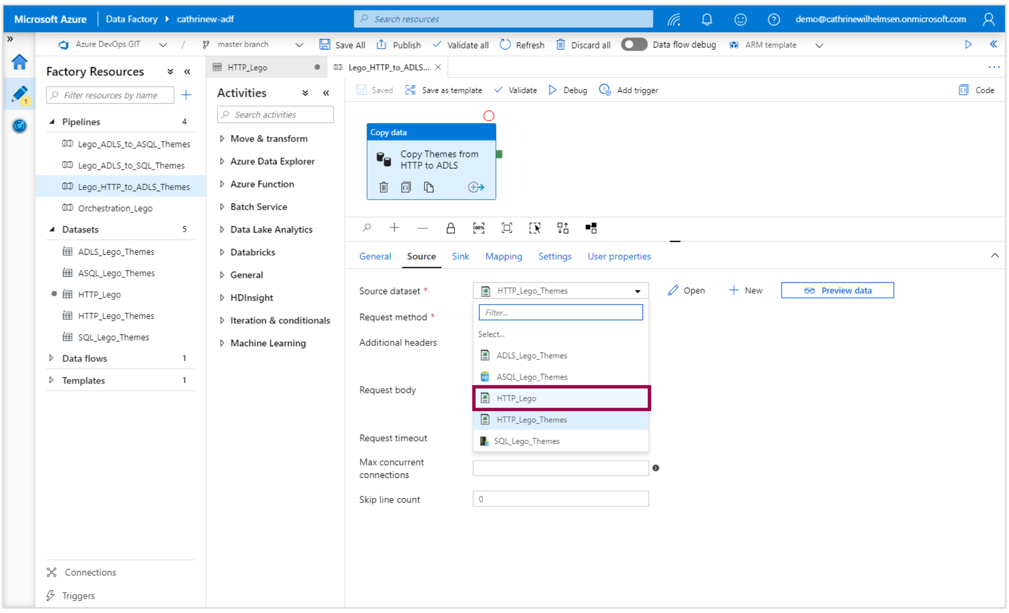


We now have a parameterized dataset, woohoo! Let's see how we can use this in a pipeline.

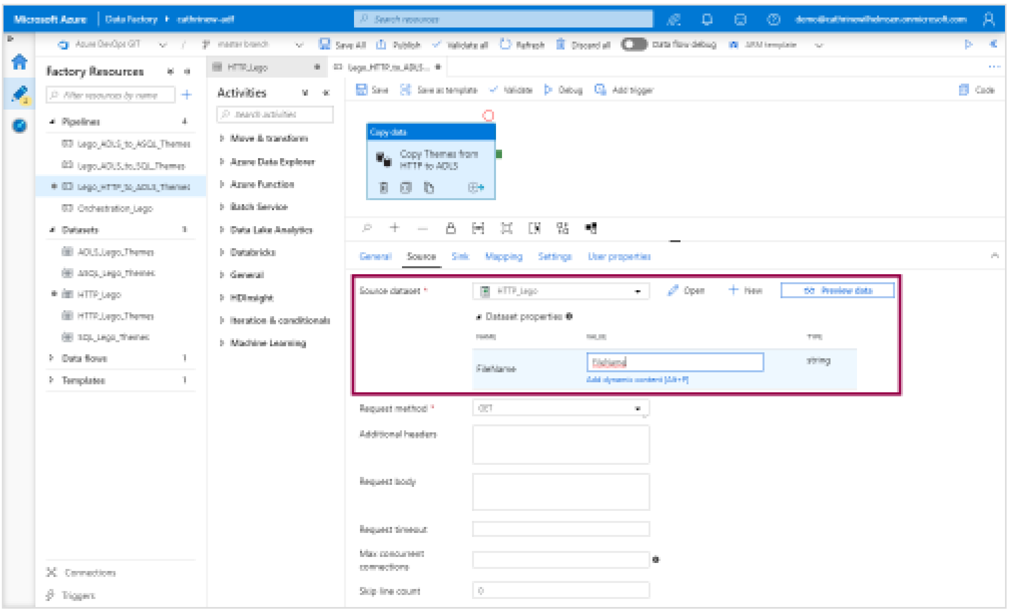
Pipeline Parameters

I have previously created a pipeline for *themes*. I'm going to change this to use the parameterized dataset instead of the *themes* dataset.

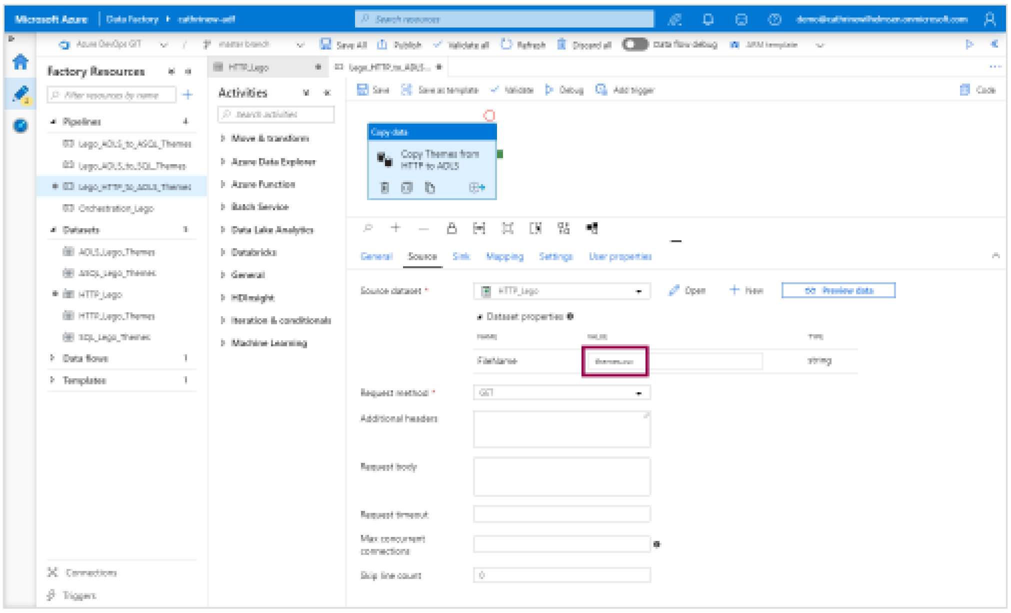
Open the **copy data activity**, and change the **source dataset**:



When we choose a parameterized dataset, the **dataset properties** will appear:



Now, we have two options. The first option is to **hardcode the dataset parameter value**:

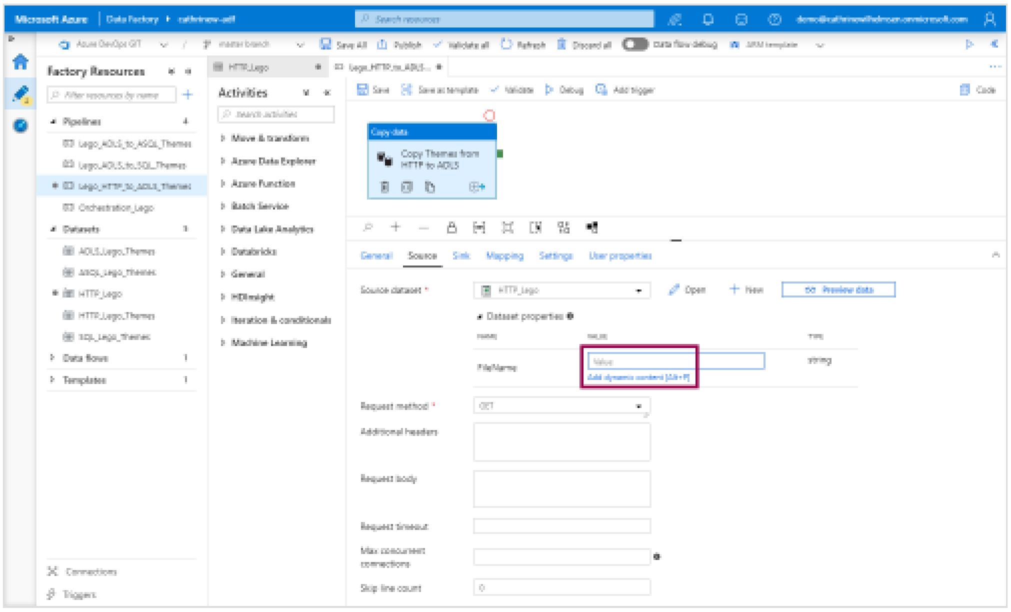


If we hardcode the dataset parameter value, we don't need to change anything else in the pipeline. The pipeline will still be for *themes* only.

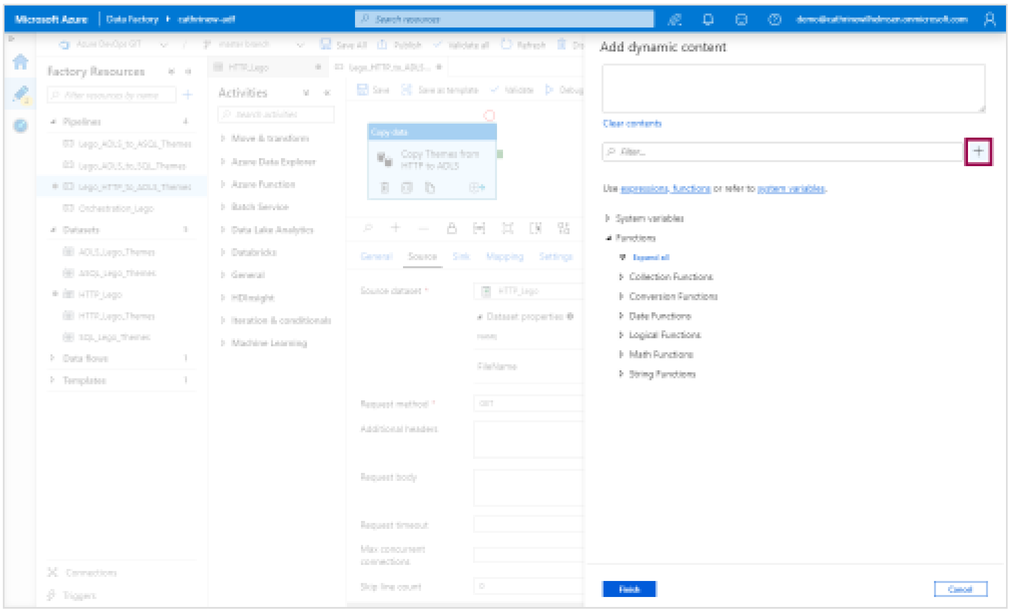
…but where's the fun in that? :D Let's change the rest of the pipeline as well!

The second option is to create a *pipeline parameter* and pass the parameter value from the pipeline into the dataset.

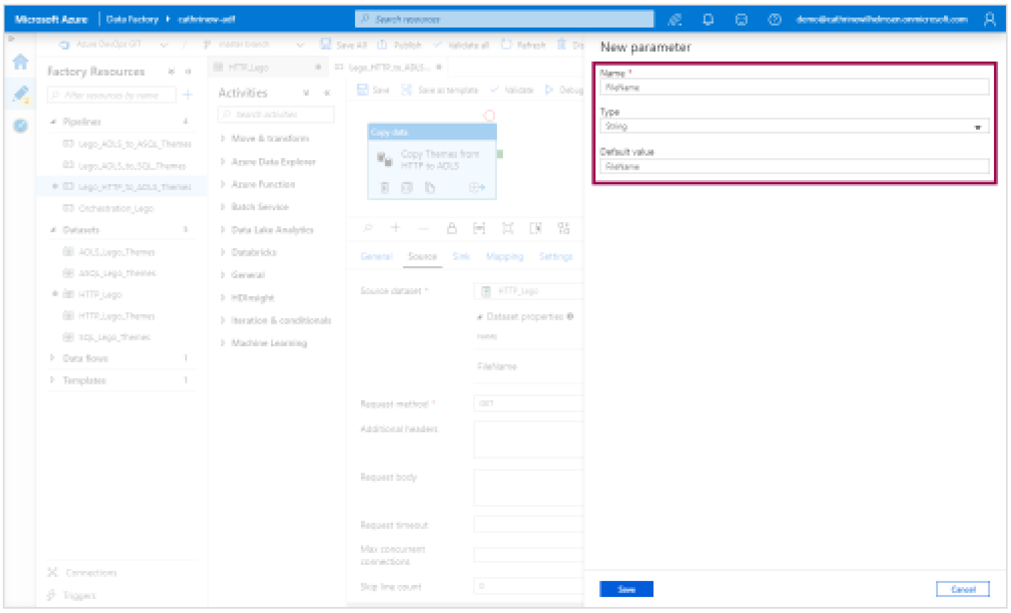
Click to open the **add dynamic content** pane:



We can create parameters from the pipeline interface, like we did for the dataset, or directly in the add dynamic content pane. There is a little **+ button** next to the filter field. Click that to create a new parameter. (*Totally* obvious, right? No, no it's not. Not at all 😂)



Create the **new parameter**:


New parameter

Name *
FileName

Type
String

Default value
FileName

Click to add the new **FileName parameter** to the dynamic content:

Notice the **@pipeline().parameters.FileName** syntax:

Changing the rest of the pipeline

To change the rest of the pipeline, we need to create a **new parameterized dataset for the sink**:

Change the **sink dataset** in the pipeline:

And **rename** the pipeline and copy data activity to something more generic:

That’s it!

…or is it?

If you are asking “*but what about the fault tolerance settings and the user properties that also use the file name?*” then I will answer “*that’s an excellent question!*” :D

There’s one problem, though… The fault tolerance setting doesn’t use “*themes.csv*“, it uses “*lego/errors/themes*“:

And the user properties contain the path information in addition to the file name:

That means that we need to rethink the *parameter value*. Instead of passing in "themes.csv", we need to pass in just "themes". Then, we can use the value as part of the filename ("themes.csv") or part of the path ("lego//themes.csv").

How do we do that?

Combining Strings

A common task in Azure Data Factory is to combine strings, for example multiple parameters, or some text and a parameter. There are two ways you can do that.

String Concatenation

The first way is to use **string concatenation**. In this case, you create an *expression* with the **concat()** *function* to combine two or more strings:

```
@concat('lego//', pipeline().parameters.FileName, '.csv')
```

(An expression starts with the **@** symbol. A function can be called within an expression.)

String Interpolation

The other way is to use **string interpolation**. This is my preferred method, as I think it's much easier to read. In this case, you create one *string* that *contains expressions* wrapped in **@{…}**:

```
lego//@{pipeline().parameters.FileName}.csv
```

No quotes or commas, just a few extra curly braces, yay :)

Using String Interpolation in Azure Data Factory

I think Azure Data Factory agrees with me that string interpolation is the way to go. Why? Well, let's try to click **auto generate** in the user properties of a pipeline that uses parameterized datasets:

Tadaaa! String interpolation. It's magic ;)

However! Since we now only want to pass in the file name, like “themes”, you need to add the “.csv” part yourself:

We also need to change the **fault tolerance settings**:

And then we need to update our datasets. In the **HTTP dataset**, change the **relative URL**:

In the **ADLS dataset**, change the **file path**:

Now you can use “themes” or “sets” or “colors” or “parts” in the pipeline, and those values will be passed into both the source and sink datasets. Cool!

Passing Parameters

But how do we use the parameter in the pipeline? Parameters can be passed into a pipeline in three ways.

You, the user, can define which parameter value to use, for example when you click **debug**:

That opens the **pipeline run** pane where you can set the parameter value:

You can set the parameter value when you **trigger now**:

That opens the **pipeline run** pane where you can set the parameter value. Notice that you have to *publish* the pipeline first, that's because we've enabled source control:

You can also add new / edit a **trigger**:

That opens the **edit trigger** pane so you can set the parameter value:

Finally, you can pass a parameter value when using the **execute pipeline** activity:

How are parameters passed?

To summarize all of this, parameters are passed in one direction. You can provide the parameter value to use manually, through triggers, or through the execute pipeline activity. Then, that parameter can be passed into the pipeline and used in an activity. Activities can pass parameters into datasets and linked services.

Summary

In this post, we looked at parameters, expressions, and functions. In the next post, we will look at variables. Then, we will cover loops and lookups.

🤓

← Templates in Azure Data Factory

Variables in Azure Data Factory →

Share?

Twitter LinkedIn Email More

Related

Copy Data Wizard in Azure Data Factory

Variables in Azure Data Factory

Overview of Azure Data Factory Components

Dec 20, 2019 Data Platform Azure Data Factory

About the Author

Cathrine Wilhelmsen is a Microsoft Data Platform MVP, BimlHero Certified Expert, Microsoft Certified Solutions Expert, international speaker, author, blogger, and chronic volunteer who loves teaching and sharing knowledge. She works as a Senior Business Intelligence Consultant at Inmeta, focusing on Azure Data and the Microsoft Data Platform. She loves sci-fi, chocolate, coffee, craft beers, ciders, cat gifs and smilies :)

Find me!

Subscribe to new posts?

E-mail

Yes, subscribe me!

Recent Posts

Sneaking back in… (2020 edition)

Keyboard shortcuts for moving text lines and windows (T-SQL Tuesday #123)

Speaking at NIC 2020

Azure Data Factory Training Day at SQLBits 2020

Personal Highlights from 2019

Popular Posts

Table Partitioning in SQL Server - The Basics

Parameters in Azure Data Factory

Preparing for and Taking Microsoft Exam DP-200 (Implementing an Azure Data Solution)

Table Partitioning in SQL Server - Partition Switching

Variables in Azure Data Factory

Top Tags

Azure Data Factory Biml Certifications Don't Repeat Yourself Microsoft Ignite Notepad++ PASS Summit Personal Precon Speaking SQLBits SQLFamily SQLHangout SQLSatOslo SQLSaturday SQL Server SSIS T-SQL Tuesday Volunteering Webinar

All Categories

Select Category

Full Archive

Select Month