Article

Using the FME Server REST API to Create Job History Reports

Jul 19, 2022 • Knowledge

Product Type
FME Server

FME Version
2022.0

Tutorial: Getting Started with the FME Server REST API (https://community.safe.com/s/article/Getting-Started-with-the-FME-Server-REST-AP) | **Previous:** Submitting Jobs via the REST API (https://community.safe.com/s/article/Submitting-a-Job-through-the-FME-Server-REST-API) | **Next:** Monitoring FME Server Job Activity using the REST API (https://community.safe.com/s/article/Monitoring-FME-Server-Job-Activity-using-the-REST-API)

Introduction

The FME Server REST API can be used for many things, from managing FME Server Engines, to submitting FME Server jobs programmatically. However, one of the most useful tasks can be generating job history reports. These reports can give you valuable information- for example, which workspace is using the most memory or CPU. This can be useful for making crucial scheduling decisions on a busy FME Server.

In this article, we will discuss the ins and outs of building a workspace to access the FME Server REST API and create reports.

This article, Monitoring FME Server Job Activity using the REST API (https://community.safe.com/s/article/Monitoring-FME-Server-Job-Activity-using-the-REST-API), shows how to use this workspace to build a dashboard in FME Server.

Requirements

- Access to FME Server
- An FME Server User account with permissions (https://docs.safe.com/fme/html/FME_Server_Documentation/WebUI/Roles.htm) to view the job history

Step-by-Step Instructions

To begin this tutorial download the workspace (AverageStatistic_Starting.fmw). This workspace is mainly complete but we will be walking through how to create some essential parts of the workspace.

Part 1: Fetch Job Statistics with Pagination

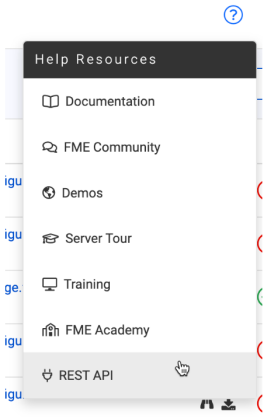
Part 1 illustrates how to fetch job history from FME Server using the FME Server REST API. This exercise uses HTTPCallers to access the FME Server REST API. For more information on how to use the HTTPCaller transformer see HTTP Requests with the HTTPCaller (https://community.safe.com/s/article/HTTP-Requests-With-The-HTTPCaller).

First, we need to figure out how many jobs were run on the FME Server using the FME Server REST API.
This is because we use the REST API to retrieve job history from FME Server and the response can only contain a maximum of 1000 jobs. If the number of jobs exceeds the limit of 1000 jobs we need to make multiple calls with the HTTPCaller. This is called pagination and can be found in most APIs.

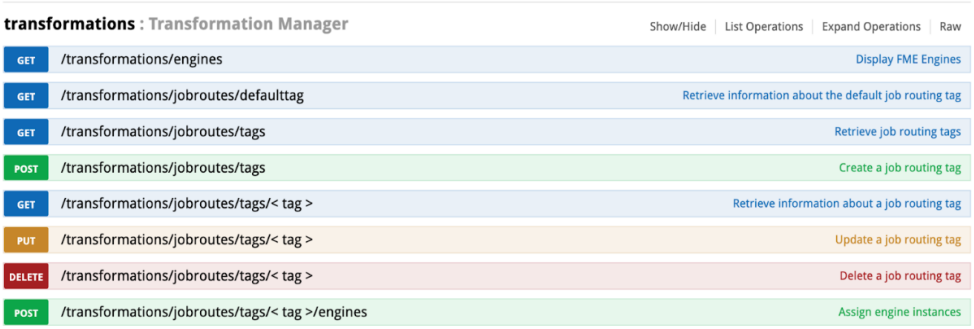
The FME Server REST API uses two parameters to handle pagination. These parameters are called limit and offset. Let's say you have a busy server with 10,000 jobs and you wanted to retrieve the full job history. For the first call, you would want to set the limit to 1000 and the offset to 0. This will retrieve the newest 1000 jobs. The next call would have to set the limit to 1000 and the offset to 1000. This would retrieve the next 1000 jobs after the first 1000.

1. Review the FME Server API Documentation

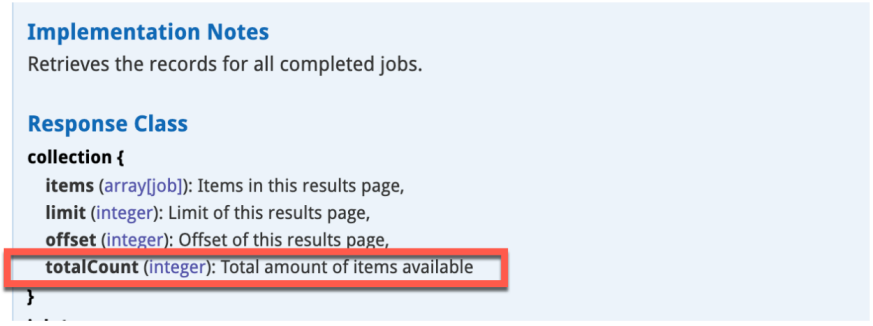
To start, review the REST API documentation (https://docs.safe.com/fme/html/FME_REST/apidoc/v3/index.html). If you access the REST API documentation from your FME Server through the Help Resources icon, you can even test REST API calls out through the web interface.



Once on the REST API documentation, select the API tab. Then, scroll to the bottom of the page and select transformations. This section shows all REST API calls related to job routing and history.



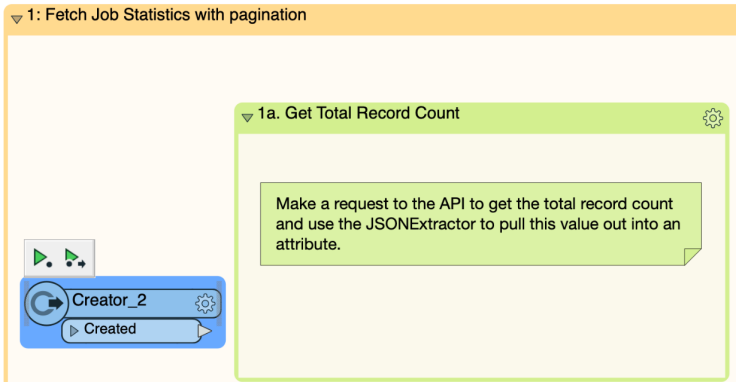
Find GET /transformations/jobs/completed. Once there, review the documentation. Notice the response is listed and it shows where to find the total count.



2. Get the total record count of all jobs run on the FME Server

Open up the workspace AverageStatistic_Starting.fmw found in the files section.
Find the bookmark 1: Fetch Job Statistics with pagination and inside it is another bookmark 1a. Get Total Record Count.

This is where we will add an HTTPCaller and a JSONExtractor to get the record count.



Add an HTTPCaller to the workspace and attach it to Creator_2.

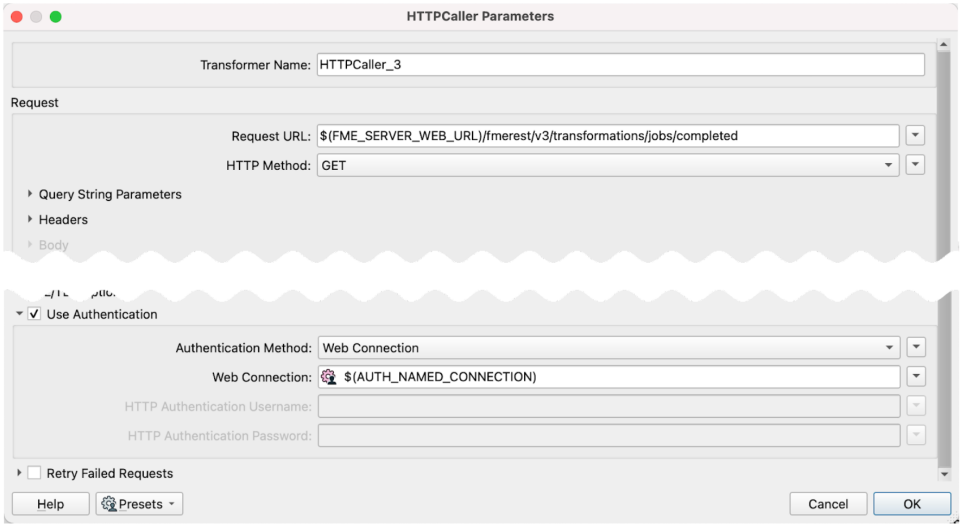
Open up the HTTPCaller and set up the following parameters.

Copy the following URL and paste it into the Request URL.

```
$(FME_SERVER_WEB_URL)/fmerest/v3/transformations/jobs/completed
```

Next, set the HTTP Method to GET.

Then, select Use Authentication. Set the Authentication Method to Web Connection. For the Web Connection, click the dropdown arrow, hover over User Parameters, then select $(AUTH_NAMED_CONNECTION)



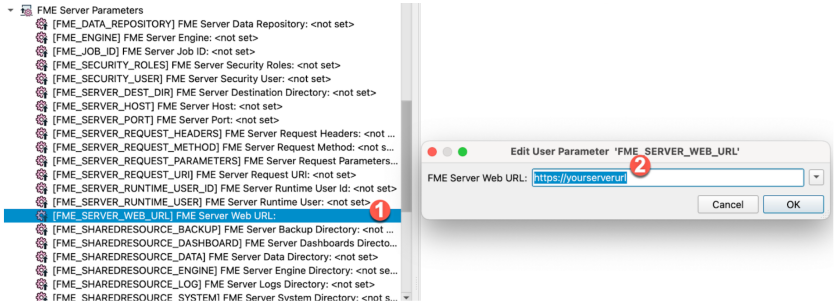
3. Set the FME Server User Parameter

In order to run this workspace on FME Desktop, you need to first set the FME Server User Parameter. The parameter we set in the HTTPCaller, allows this workspace to run on any FME Server, as the Server URL will be set automatically in FME Server.

```
$(FME_SERVER_WEB_URL)
```

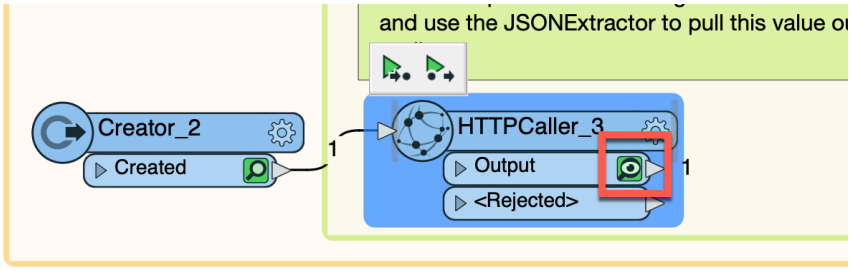
However, this means the workspace won't run on Desktop without setting up the parameter. In the navigator panel, find the FME Server Parameter, then find the FME Server Web URL parameter. Double click on the parameter this will open up a text box to add your FME Server URL. ***Make sure to not include a final slash at the end of the URL.***

Click OK to exit the dialog.

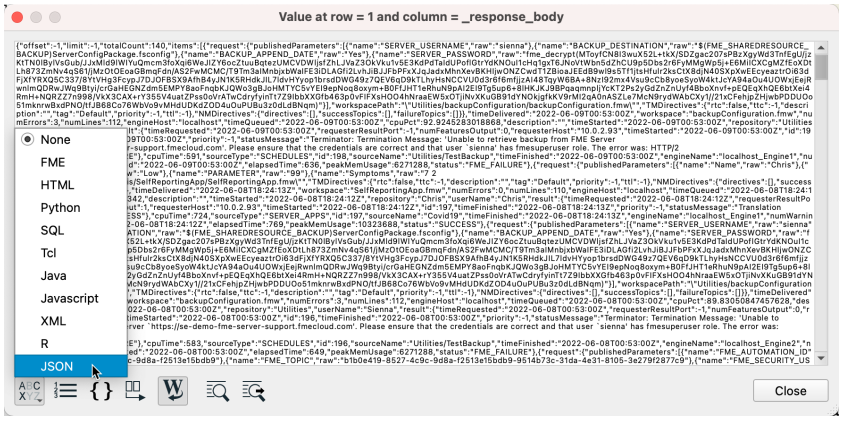


4. Run the FME Workspace

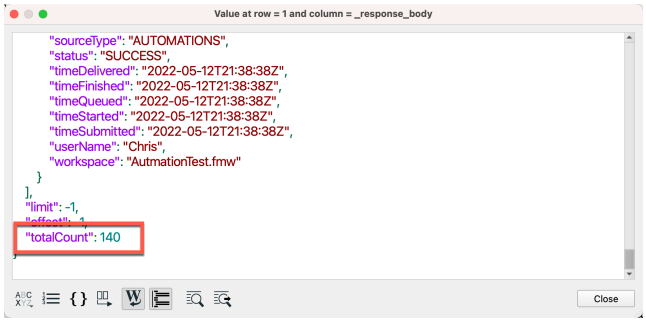
Run the Workspace with Feature Caching Enabled. Review the attribute from the output called _response_body.



To get a better view of the JSON, click the ABCXYZ button, then select JSON.



Scroll down to the bottom of the page to view the totalCount from your server.



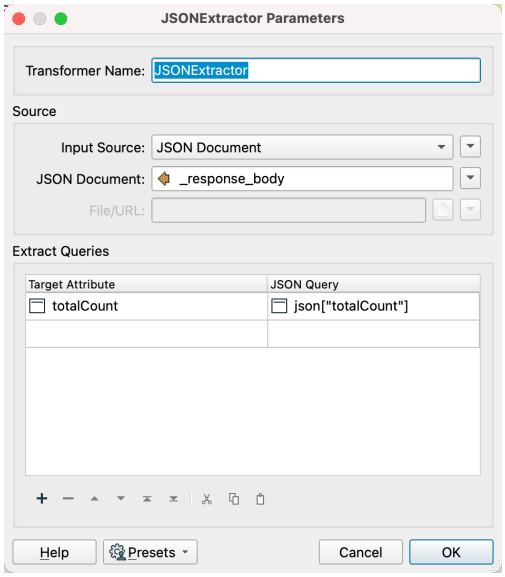
5. Get the totalCount from the JSON response

Add a JSONExtractor to the canvas. Connect the JSON Extractor to the output port of the HTTPCaller. Then, connect the Evaluated port from the JSONExtractor to the ExpressionEvaluater_3.

Open the JSONExtractor to modify the parameters.

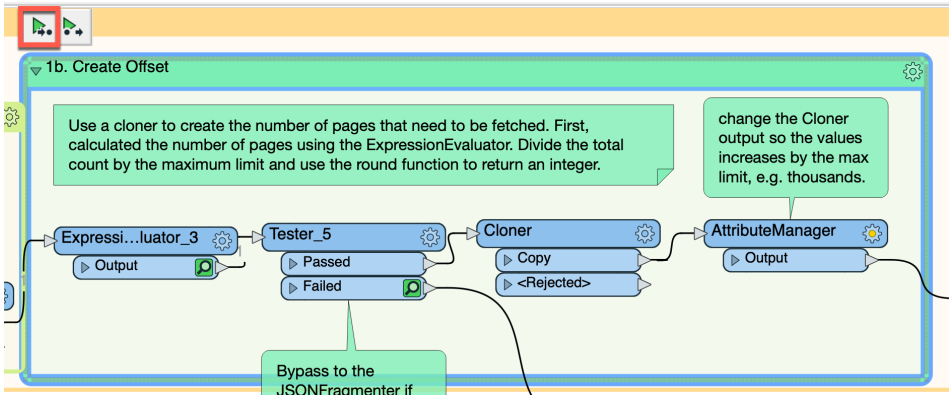
Click the drop-down beside JSON Document, hover over Attribute Value, then select _response_body.

Under Target Attribute, write totalCount. Then, under JSON Query, write json["totalCount"]



Run the workspace again and note that a new attribute called totalCount is created.

6. Run the 1b. Create Offset Bookmark



Click the bookmark and select Run to Contained. This bookmark will create the requests needed to fetch all of the job records
If the total count exceeds over 1000 records, then we need to submit multiple calls to get the results.

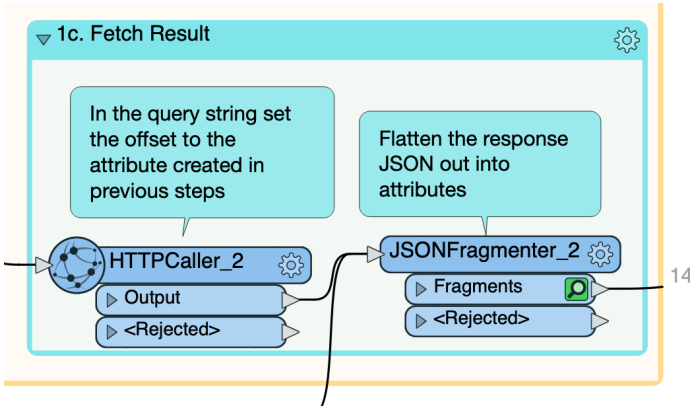
The ExpressionEvaluator is used to determine what the offset will be. The total count is divided by 1000 to determine the number of pages we'd need to fetch. This number is then rounded to ensure there are no decimal places.

Next, a Tester is used. If the first call retrieved less than 1000 results then we do not need to make additional calls so we bypass the Cloner.

The Cloner is used to create multiple features based on the offset number. For instance, if the offset is set to 3, we will create 3 copies of a feature with a copy number (a count of the copies) we call offset.

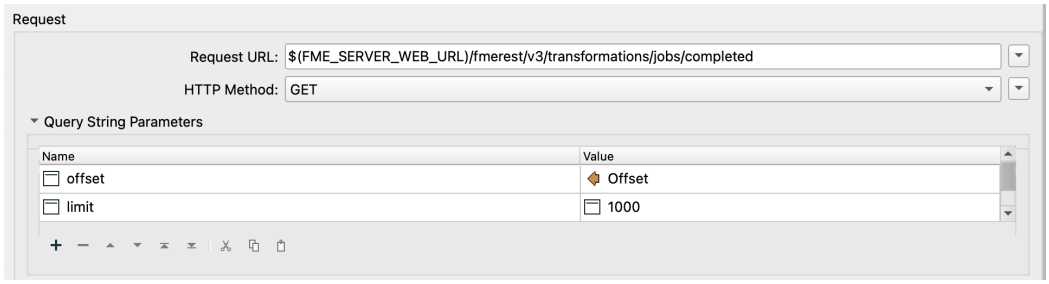
Next, the AttributeManager is used to increase the offset by 1000, since we want to grab the next 1000 records instead of the next record.

7. Run the 1c. Fetch Results Bookmark

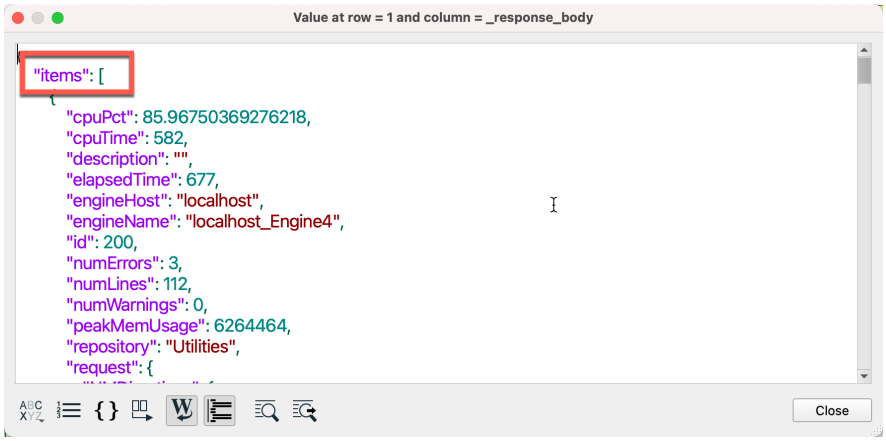


14

Here, the HTTPCaller is used to fetch the remaining results, if there are any. The offset is used to tell the API where to start fetching the next 1000 records.



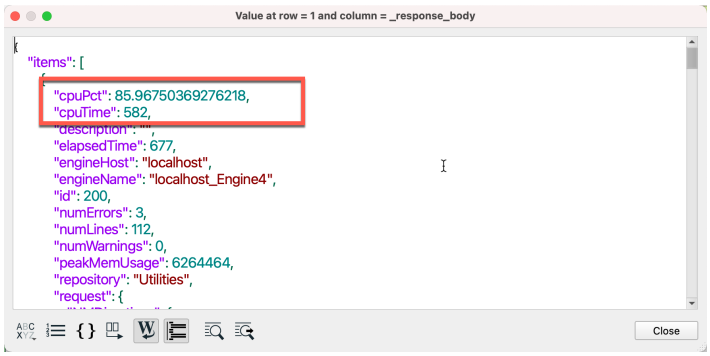
Next, the JSONFragmenter is used to parse the results. Take a look at the original _response_body from the Output of the HTTPCaller. Notice the first item is “items” and all the information about the job is held within this item.



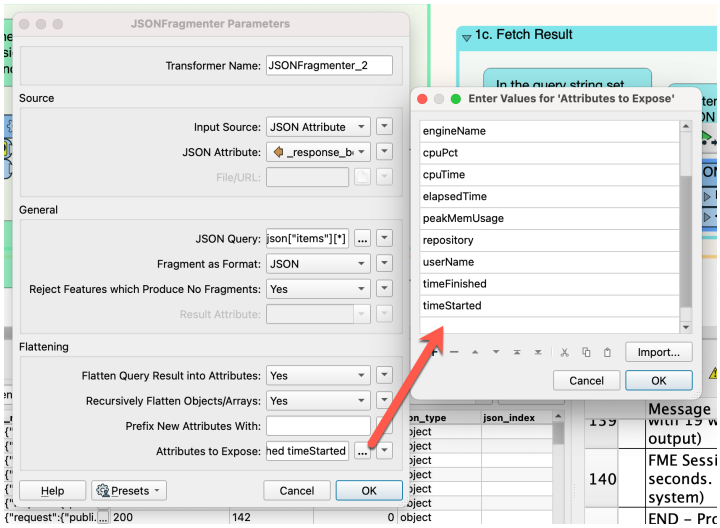
Open up the JSONFragmenter_2, the JSON query is set to *json["items"][*]*

This action allows the user to expose all attributes inside of the item's tag.

Next, the attributes to expose section will now expose the attributes listed. These attributes can be found in the response body.



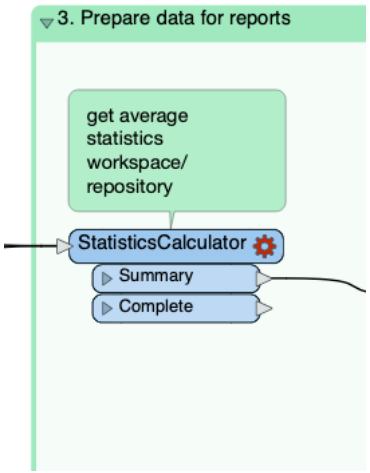
These attributes are manually entered in the “Attribute to Expose” parameter, or these can be imported if you have a file saved with the JSON using the Import… button.



Part 2: Prepare data and create reports

Currently, the data is retrieved by the REST API, but it is not in a format conducive to creating informative reports. The stars of this next section are the StatisticCalculator and the HTMLReportGenerator. The StatisticsCalculator calculates statistics based on a designated attribute or set of attributes. The HTMLReportGenerator is used to take this data and turn it into an HTML report.

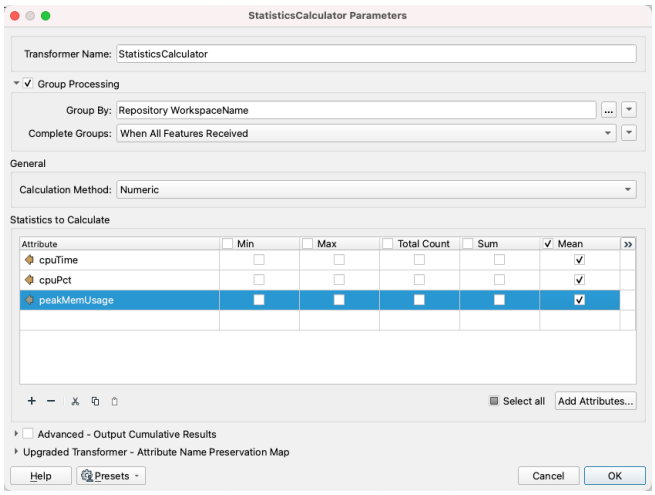
1. Configure the StatisticsCalculator



Open the StatisticsCalculator and set it up to calculate the following attributes.

First, enable Group Processing. Select Group by and check Repository and WorkspaceName. By using the Group by option, the statistics are calculated for the group. In our case statistics will be calculated per workspace. Instead of the statistics being calculated for all the workspaces together.

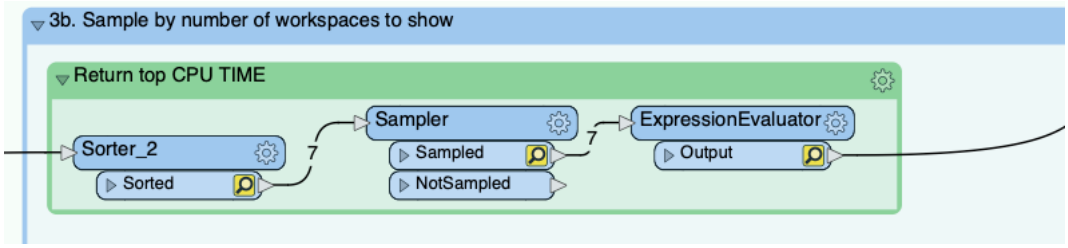
Next, set up the Statistics to Calculate section. Select the dropdown under Attributes and select cpuTime. Then, click the Mean checkbox. Do the same for the cpuPct, peakMemUsage attributes.



Please note that you can calculate multiple statistics for an attribute. For example, on cpuTime, you could calculate the Mean and the Total Count.

2. Sort and Calculate the CPU Time

Next, review the bookmark 3b. Sample by number of workspaces to show. Here, we will only review one of the bookmarks, because the rest are similar.



Take a look at the Return Top CPU Time bookmark. Here the Sorter is used to sort the longest-running workspaces in descending order.

Then, the Sampler is used to fetch the number of workspaces specified by the user parameter SamplingRate. This enables the user to select how many workspaces they'd like to see in the report.

Next, the ExpressionEvalutor is used to convert the CPU Time which is currently in milliseconds to seconds. This result is then rounded.

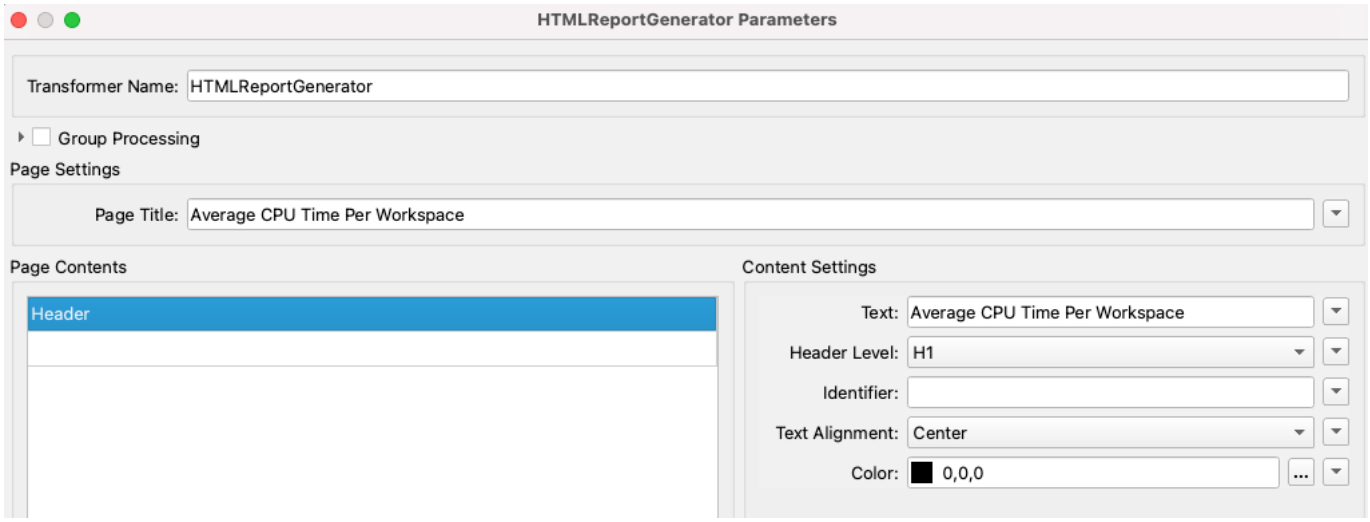
After this, the data is ready to go into the HTMLReportGenerator.

3. Configure the HTMLReportGenerator

Take a look at the bookmark *4a. Create Average CPU Time HTML Report.* Open up the HTMLReportGenerator and configure the following settings.

First, set the page title to "Average CPU Time Per Workspace".

Then, under Page Contents, select Header. Click in the Content Settings on the right to view the Header settings. Set the Text to "Average CPU Time Per Workspace". Set the header level to H1 and the text alignment to Center.

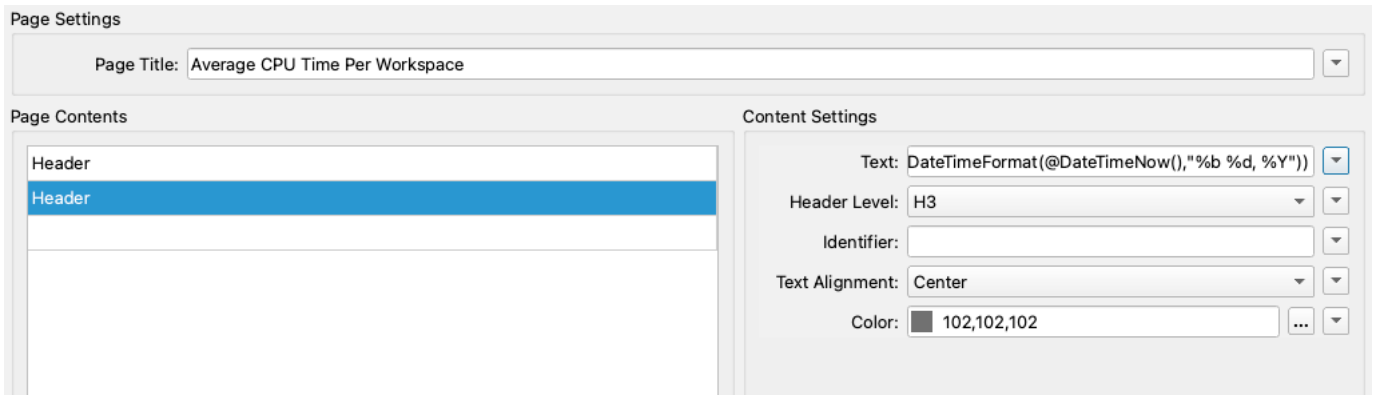


Next, under Page Contents, add another Header. In the Content Settings set the Text to:

```
Top $(SamplingRate) Largest CPU Workspaces (@DateTimeFormat(@DateTimeAdd(@DateTimeNow(),-P7D),"%b %d, %Y")
- @DateTimeFormat(@DateTimeNow(),"%b %d, %Y"))
```

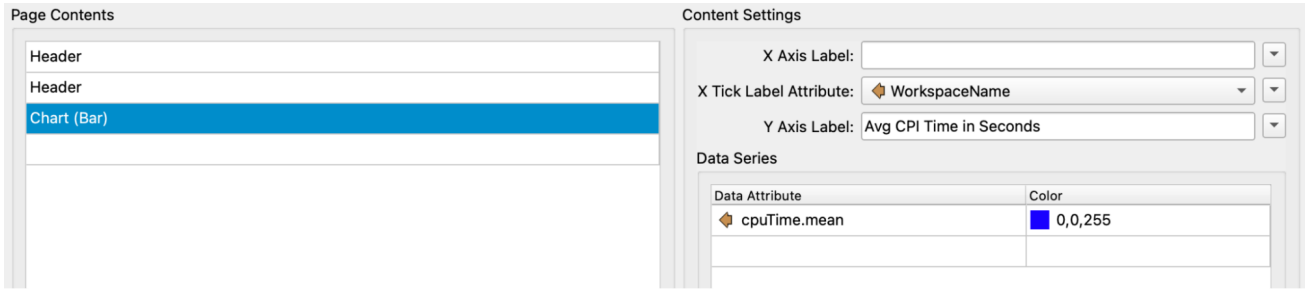
Notice how the Datetime functions are used within the text editor. These functions are used to get a timestamp and format so that it is easier to read. This can also be handled in our DateTime transformers, if you find the Datetime functions confusing.

Set this Header Level to H3, the Text Alignment to Center, and the Color to 102,102,102



Next, add a new Page Content and set it to Chart (Bar). On the right-hand side, leave the X Axis Label blank, set the X Tick Label Attribute to WorkspaceName, and set the Y Axis Label to the Average CPI Time in Seconds.

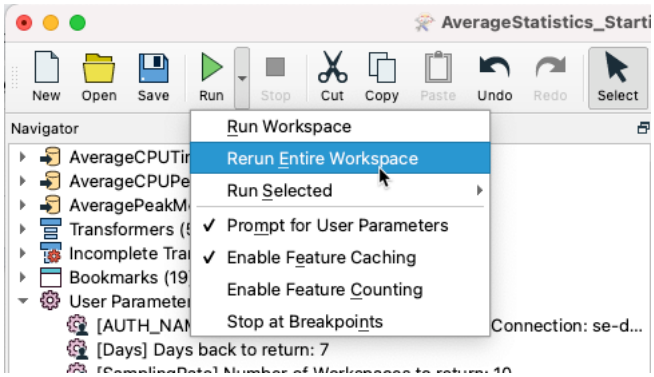
In the Data Series, set the Data Attribute to cpuTime.mean and Color to 0,0,255



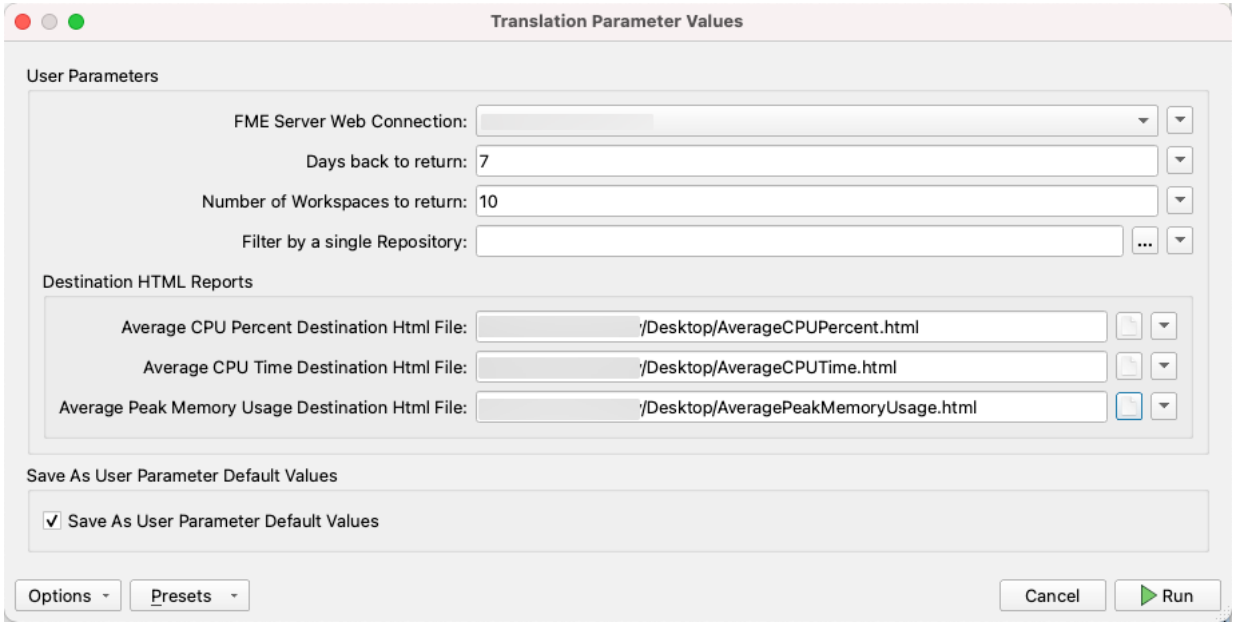
Note, StringReplacers are used after the HTMLReportGenerator to change the styling however, this isn't necessary. It's just a personal adjustment.

4. Run the Workspace and Review the Results

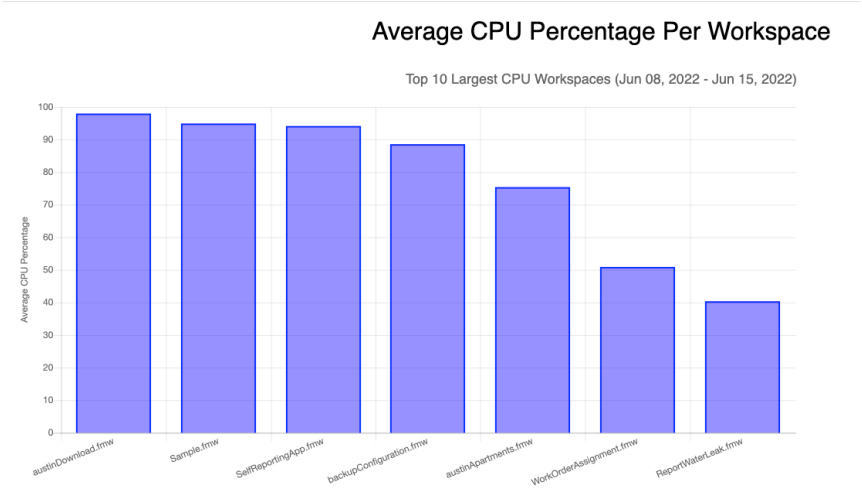
Finally, click the dropdown by the Run Button and select Rerun Entire Workspace.



This will bring up a prompt to set the parameter values. Make sure to set your FME Server Web Connection and select locations for the HTML Reports. Then, select Run.



Now, review the HTML files created! Below is an example of a report created through this workspace.



Congratulations! You have completed this exercise. To continue with this series, see Monitoring FME Server Job Activity using the REST API (https://community.safe.com/s/article/Monitoring-FME-Server-Job-Activity-using-the-REST-API) to learn how to take this workspace and create a dashboard from it.

First Published Date
7/6/2022, 9:04 PM

Last Published Date
7/19/2022, 10:57 PM

Sort by:
Latest Posts

siennaatsafe (/s/profile/0050c00000CzqRuAAJ) (Employee) published a new version of this Knowledge.
July 19, 2022 at 10:57 PM (/s/feed/0D54Q00009fDii9SAC)

Like Comment

Log In to Comment

siennaatsafe (/s/profile/0050c00000CzqRuAAJ) (Employee) published this new Knowledge.
July 6, 2022 at 9:04 PM (/s/feed/0D54Q00009dLOIkSAO)

Like Comment

Log In to Comment

Follow

Files (1) (/s/relatedlist/ka14Q000001DWtNQAW/AttachedContentDocuments)

Chat with Us

AverageStatistics_Starting
Jul 18, 2022 • 2.7MB • fmw

View All

(/s/relatedlist/ka14Q000001DWtNQAW/AttachedContentDocuments)

Related Articles

Monitoring FME Server Job Activity using the REST API (/s/article/Monitoring-FME-Server-Job-Activity-using-the-REST-API)

FME Server REST API /healthcheck Endpoint (/s/article/fme-server-rest-api-healthcheck-endpoint)

FME Server Troubleshooting: REST API (/s/article/fme-server-troubleshooting-rest-api)

Tips for Working With the FME Server REST API in FME Workbench (/s/article/Tips-for-Working-With-the-FME-Server-REST-API-in-FME-Workbench)

Authorization in the FME Server REST API | Token Management (/s/article/token-management-in-fme-server)

Land Acknowledgement —

Safe Software respectfully acknowledges that we live, learn and work on the traditional and unceded territories of the Kwantlen, Katzie, and Semiahmoo First Nations.