

Document AI Parser Result Merger(External)

Table of Contents

Disclaimer	1
Objective	2
Case 1: Different documents, parser results json merger (Default).	2
Case 2: Same document, different parsers json merger(Added functionality).	2
Prerequisites	2
Workflow overview:	2
Script walkthrough	3
1. Config file Creation	3
2. Input Details : Entering Project details in Config files:	4
3. Run the below code.	5
4. Output	18
CASE - 1 Output	18
CASE - 2 Output	19

Disclaimer

This tool is not supported by the Google engineering team or product team. It is provided and supported on a best-effort basis by the **DocAI Incubator Team**. No guarantees of performance are implied.

Objective

Document AI Parser Result Merger is a tool built using Python programming language. Its purpose is to address the issue of merging the two or more resultant json files of Document AI processors. This document highlights the working of the tool(script) and its requirements. The documents usually contain multiple pages. There are 2 use cases by which this solution can be operated.

Case 1: Different documents, parser results json merger (Default).

- Case 1 deals when we are using two or multiple parser output Jsons are from **different documents**
- To Enable this case the flag should be '1'

Case 2: Same document, different parsers json merger(Added functionality).

- Case 2 deals when we are using two or multiple parser outputs from the **same document**.
- To Enable this case the flag should be '2'

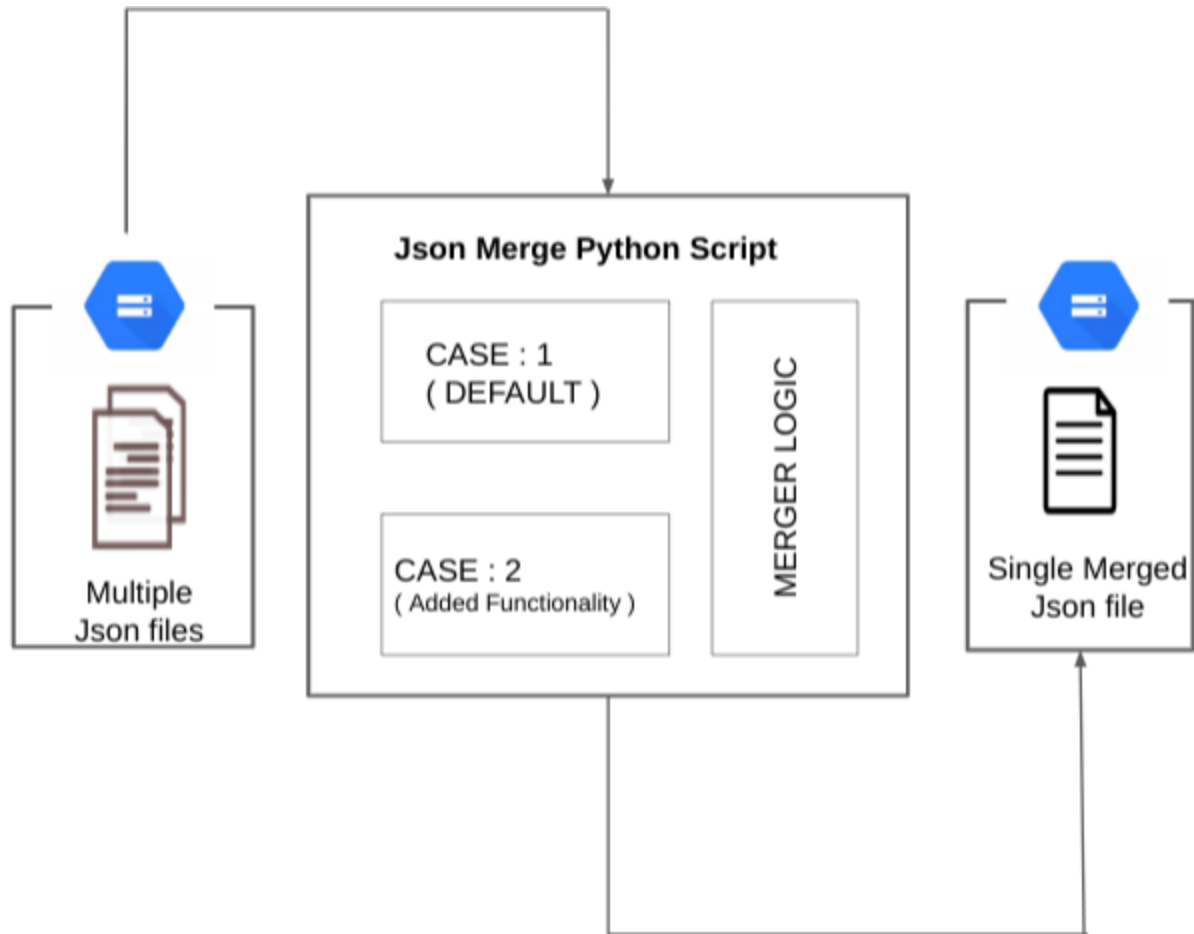
Prerequisites

This tool requires the following services:

- Google Jupyter Notebook or Colab.
- Google Cloud Storage
- DocumentAI processor and JSON files

Google Jupyter Notebook or Colab is used for running the python notebook file. Cloud Storage Buckets have the input files to this script. The multiple input files are the json files which are the result of a Document AI processor (for eg., Bank Statement Parser). These json files include multiple pages in its document. After the script executes, the output file is a single merged json file stored in the output bucket path.

Workflow overview:



The above diagram shows the flow diagram of the tool. As highlighted there are input and output GCP buckets and there is a python script which processes the request. The input bucket holds the multiple json files which need to be merged into a single file and this is achieved by the python script. This script accepts the input json files and prompts users to switch between the default case-1 or the case-2 mode as highlighted in the previous sections. Finally there is an output GCP bucket to store the single merged file.

Script walkthrough

Insights and details about the script are explained in detail as follows.

1. Config file Creation

Run the below code and create a config.ini file for providing input.

Python

```
import configparser
config = configparser.ConfigParser()
# Add the structure to the file we will create
config.add_section('Parameters')
config.set('Parameters', 'project_id', 'xxxx-xxxx-xxxx')
config.set('Parameters', 'Input_Multiple_jsons_URI', 'gs:///')
config.set('Parameters', 'Output_Multiple_jsons_URI', 'gs:///')
config.set('Parameters', 'Name_output_Json', 'merged_json')
config.set('Parameters', 'merger_type_flag(1-for different
docs,2-same doc)', '1')

# Write the new structure to the new file
with open(r"configfile.ini", 'w') as configfile:
    config.write(configfile)
```

2. Input Details : Entering Project details in Config files:

```
1 [Parameters]
2 project_id = rand-automl-project
3 input_multiple_jsons_uri = gs://json_doc_merge/
4 output_multiple_jsons_uri = gs://json_doc_merge/output/
5 name_output_json = merged_json.json
6 merger_type_flag(1-for different docs,2-same doc) = 1
```

Once the config.ini file is created with the above step 1 , open the config.ini file and enter the input details specific to your project and GCP bucket paths. As shown in the diagram above, the following parameters are to be entered.

- **project_id:** provide your GCP project ID
- **input_multiple_jsons_uri:** provide the uri link of folder containing the input files
- **output_multiple_jsons_uri:** provide the folder name of the output file which gets generated post execution of the script.
- **Name_output_json:** enter a name for the generated file which is saved in the output bucket.

- **merger_type_flag(1-for different docs,2-same doc)** : based on user need, values 1 or 2 can be provided as mentioned in the earlier part of this document.
- ❖ Case 1 deals when we are using two or multiple parser output Jsons are from **different documents**
- ❖ Case 2 deals when we are using two or multiple parser outputs from the **same document**.

3. Run the below code.

Use the below code and Run all the cells (Update the Path parameter if it is not available in the current working directory)

Python

```
#importing Libraries
import os, json, re, gc
import gcsfs
from pprint import pprint
import urllib.request
import pandas as pd
from datetime import datetime
from google.api_core.client_options import ClientOptions
from google.api_core.exceptions import InternalServerError,
RetryError
from google.cloud import storage
import configparser

#Getting Input from config file
Path= "configfile.ini" #Enter the path of config file

config = configparser.ConfigParser()
config.read(Path)
project_id=config.get('Parameters','project_id')
input_multiple_jsons_uri =
config.get('Parameters','input_multiple_jsons_uri')
JSON_DIRECTORY_PATH_OUTPUT =
config.get('Parameters','Output_Multiple_jsons_URI')
output_file_name = config.get('Parameters','Name_output_Json')
```

```
merger_type_flag=config.get('Parameters','merger_type_flag(1-for
different docs,2-same doc)')
```

```
#Functions
```

```
### CASE - 1
```

```
def merger(doc_first, doc_second):
```

```
    doc_merged={}
```

```
    ### Entities ###
```

```
    for entity in doc_second['entities']:
```

```
        try:
```

```
            #print("\n\n+++++++ PAGE ANCHOR ++++++")
```

```
            for x in
```

```
range(0,len(entity['pageAnchor']['pageRefs'])):
```

```
            try:
```

```
                entity['pageAnchor']['pageRefs'][x]['page'] =
```

```
str(int(entity['pageAnchor']['pageRefs'][x]['page']) +
```

```
len(doc_first['pages']))
```

```
            except:
```

```
                entity['pageAnchor']['pageRefs'][x]['page'] =
```

```
str(len(doc_first['pages']))
```

```
        except:pass
```

```
    #print("--- Properties ---")
```

```
    try:
```

```
        for x in range(0,len(entity['properties'])):
```

```
            for xx in
```

```
range(0,len(entity['properties'][x]['pageAnchor']['pageRefs'])):
```

```
                try:
```

```

entity['properties'][x]['pageAnchor']['pageRefs'][xx]['page'] =
int(entity['properties'][x]['pageAnchor']['pageRefs'][xx]['page']
) + len(doc_first['pages'])
        except:

entity['properties'][x]['pageAnchor']['pageRefs'][xx]['page'] =
len(doc_first['pages'])
        except:pass

        #print("+++++++ TEXT ANCHOR ++++++")

        try:
            textAnchor = entity['textAnchor']
            for y in range(0,
len(entity['textAnchor']['textSegments'])):

entity['textAnchor']['textSegments'][y]['endIndex'] =
int(entity['textAnchor']['textSegments'][y]['endIndex']) +
len(doc_first['text'])
                try:

entity['textAnchor']['textSegments'][y]['startIndex'] =
int(entity['textAnchor']['textSegments'][y]['startIndex']) +
len(doc_first['text'])
                    except: # if startIndex is absent

entity['textAnchor']['textSegments'][y]['startIndex'] =
len(doc_first['text'])
                except:pass

        #print("--- Properties ---")

        try:
            for y in range(0,len(entity['properties'])):

```

```

        for yy in
range(0, len(entity['properties'][y]['textAnchor']['textSegments']
)):

entity['properties'][y]['textAnchor']['textSegments'][yy]['endInd
ex'] =
int(entity['properties'][y]['textAnchor']['textSegments'][yy]['en
dIndex']) + len(doc_first['text'])
        try:

entity['properties'][y]['textAnchor']['textSegments'][yy]['startI
ndex'] =
int(entity['properties'][y]['textAnchor']['textSegments'][yy]['st
artIndex']) + len(doc_first['text'])
        except: # if startIndex is absent

entity['properties'][y]['textAnchor']['textSegments'][yy]['startI
ndex'] = len(doc_first['text'])
        except:pass


doc_merged['entities'] = doc_first['entities'] +
doc_second['entities']


### Page
### Page No increment in second doc
for pg in doc_second['pages']:
    print(pg['pageNumber'])
    pg['pageNumber'] = int(pg['pageNumber']) +
len(doc_first['pages'])
    print('\t', pg['pageNumber'])

```



```

    ### Page
    ### page . blocks . layout . textanchor . textsegment
    for pg in range(0, len(doc_second['pages'])):
        for pg_ in
range(0, len(doc_second['pages'][pg]['blocks'])):
            for pg_textSegment in
range(0, len(doc_second['pages'][pg]['blocks'][pg_]['layout']['textAnchor']['textSegments'])):

doc_second['pages'][pg]['blocks'][pg_]['layout']['textAnchor']['textSegments'][pg_textSegment]['endIndex'] =
int(doc_second['pages'][pg]['blocks'][pg_]['layout']['textAnchor']['textSegments'][pg_textSegment]['endIndex']) +
len(doc_first['text'])
                try:

doc_second['pages'][pg]['blocks'][pg_]['layout']['textAnchor']['textSegments'][pg_textSegment]['startIndex'] =
int(doc_second['pages'][pg]['blocks'][pg_]['layout']['textAnchor']['textSegments'][pg_textSegment]['startIndex']) +
len(doc_first['text'])
                    except:

doc_second['pages'][pg]['blocks'][pg_]['layout']['textAnchor']['textSegments'][pg_textSegment]['startIndex'] =
len(doc_first['text'])

    ### page . layout . textanchor . textsegment
    for pg in range(0, len(doc_second['pages'])):
        #print("----")

#print(doc_second['pages'][pg]['layout']['textAnchor']['textSegments'])

```

```

        for pg_textSegment in range(0,
len(doc_second['pages'][pg]['layout']['textAnchor']['textSegments
']))):

```

```

doc_second['pages'][pg]['layout']['textAnchor']['textSegments'][p
g_textSegment]['endIndex'] =
int(doc_second['pages'][pg]['layout']['textAnchor']['textSegments
'][pg_textSegment]['endIndex']) + len(doc_first['text'])
        try:

```

```

doc_second['pages'][pg]['layout']['textAnchor']['textSegments'][p
g_textSegment]['startIndex'] =
int(doc_second['pages'][pg]['layout']['textAnchor']['textSegments
'][pg_textSegment]['startIndex']) + len(doc_first['text'])
        except:

```

```

doc_second['pages'][pg]['layout']['textAnchor']['textSegments'][p
g_textSegment]['startIndex'] = len(doc_first['text'])

```

```

    ### page . lines . layout . textanchor . textsegment
    for pg in range(0,len(doc_second['pages']))):

```

```

        for pg_line in
range(0,len(doc_second['pages'][pg]['lines']))):
            for pg_textSegment in
range(0,len(doc_second['pages'][pg]['lines'][pg_line]['layout']['
textAnchor']['textSegments']))):

```

```

doc_second['pages'][pg]['lines'][pg_line]['layout']['textAnchor']
['textSegments'][pg_textSegment]['endIndex'] =
int(doc_second['pages'][pg]['lines'][pg_line]['layout']['textAnch
or']['textSegments'][pg_textSegment]['endIndex']) +
len(doc_first['text'])
            try:

```

```

doc_second['pages'][pg]['lines'][pg_line]['layout']['textAnchor']
['textSegments'][pg_textSegment]['startIndex'] =
int(doc_second['pages'][pg]['lines'][pg_line]['layout']['textAnchor']
['textSegments'][pg_textSegment]['startIndex']) +
len(doc_first['text'])
except:

```

```

doc_second['pages'][pg]['lines'][pg_line]['layout']['textAnchor']
['textSegments'][pg_textSegment]['startIndex'] =
len(doc_first['text'])

```

```

### page . paragraph . layout . textanchor . textsegment
for pg in range(0, len(doc_second['pages'])):
    for pg_paragraph in
range(0, len(doc_second['pages'][pg]['paragraphs'])):
        for pg_textSegment in
range(0, len(doc_second['pages'][pg]['paragraphs'][pg_paragraph]['
layout']['textAnchor']['textSegments'])):

```

```

doc_second['pages'][pg]['paragraphs'][pg_paragraph]['layout']['te
xtAnchor']['textSegments'][pg_textSegment]['endIndex'] =
int(doc_second['pages'][pg]['paragraphs'][pg_paragraph]['layout']
['textAnchor']['textSegments'][pg_textSegment]['endIndex']) +
len(doc_first['text'])
try:

```

```

doc_second['pages'][pg]['paragraphs'][pg_paragraph]['layout']['te
xtAnchor']['textSegments'][pg_textSegment]['startIndex'] =
int(doc_second['pages'][pg]['paragraphs'][pg_paragraph]['layout']
['textAnchor']['textSegments'][pg_textSegment]['startIndex']) +
len(doc_first['text'])
except:

```

```
doc_second['pages'][pg]['paragraphs'][pg_paragraph]['layout']['textAnchor']['textSegments'][pg_textSegment]['startIndex'] = len(doc_first['text'])
```

```
    ### page . tokens . layout . textanchor . textsegment
    for pg in range(0, len(doc_second['pages'])):
        for pg_token in
range(0, len(doc_second['pages'][pg]['tokens'])):
            for pg_textSegment in
range(0, len(doc_second['pages'][pg]['tokens'][pg_token]['layout']
['textAnchor']['textSegments'])):
```

```
doc_second['pages'][pg]['tokens'][pg_token]['layout']['textAnchor']
['textSegments'][pg_textSegment]['endIndex'] =
int(doc_second['pages'][pg]['tokens'][pg_token]['layout']['textAnchor']
['textSegments'][pg_textSegment]['endIndex']) +
len(doc_first['text'])
        try:
```

```
doc_second['pages'][pg]['tokens'][pg_token]['layout']['textAnchor']
['textSegments'][pg_textSegment]['startIndex'] =
int(doc_second['pages'][pg]['tokens'][pg_token]['layout']['textAnchor']
['textSegments'][pg_textSegment]['startIndex']) +
len(doc_first['text'])
        except:
```

```
doc_second['pages'][pg]['tokens'][pg_token]['layout']['textAnchor']
['textSegments'][pg_textSegment]['startIndex'] =
len(doc_first['text'])
```

```
doc_merged['pages'] = doc_first['pages'] +  
doc_second['pages']
```

```
### Text
```

```
doc_merged['text'] = doc_first['text'] + doc_second['text']
```

```
### shardInfo & uri
```

```
if 'shardInfo' in doc_first:
```

```
    doc_merged['shardInfo'] = doc_first['shardInfo']
```

```
if 'uri' in doc_first:
```

```
    doc_merged['uri'] = doc_first['uri']
```

```
return doc_merged
```

```
### CASE -2
```

```
def SameDocDiffParser_merger(doc_first, doc_second):
```

```
    doc_first['entities'] = doc_first['entities'] +
```

```
doc_second['entities']
```

```
    doc_merged = doc_first
```

```
    return doc_merged
```

```
def file_names(file_path):
```

```
    """This Function will load the bucket and get the list of  
files
```

```
    in the gs path given
```

```
    args: gs path
```

```
    output: file names as list and dictionary with file names as  
keys and file path as values"""
```

```
    bucket=file_path.split("/")[2]
```

```
    file_names_list=[]
```

```
    file_dict={}
```

```
    storage_client = storage.Client()
```

```
    source_bucket = storage_client.get_bucket(bucket)
```

```

        filenames = [filename.name for filename in
list(source_bucket.list_blobs(prefix=(( '/' ).join(file_path.split(
 '/' )[3: ])))))]
    for i in range(len(filenames)):
        x=filenames[i].split(' / ')[-1]
        if x is not "":
            file_names_list.append(x)
            file_dict[x]=filenames[i]
    return file_names_list,file_dict

```

```

file_names_list, file_dict = file_names(input_multiple_jsons_uri)

```

```

#prints filenames in list
#file_names_list

```

```

#prints path in dict
#file_dict

```

```

input_bucket_files = []
for fldrFile in file_names_list:
    if(fldrFile.endswith('.json')):
        print(fldrFile)
        input_bucket_files.append(fldrFile)
print(input_bucket_files)

```

```

storage_client = storage.Client()
source_bucket =
storage_client.get_bucket(input_multiple_jsons_uri.split(' / ')[2])

```

```

fs=gcsfs.GCSFileSystem(project=project_id)

```

```

#storage_client = storage.Client()
#input_bucket_name      = input_multiple_jsons_uri.split("/")[2]

```

```

#bucket = storage_client.bucket(input_bucket_name)
#fs=gcsfs.GCSFileSystem(project=project_id)
#input_bucket_files = []
#for fldr in bucket.list_blobs():
#    if(fldr.name.endswith('.json')):
#        print(fldr.name)
#        input_bucket_files.append(fldr.name)

### CASE - 1
if(merger_type_flag == '1'):
    print('>>> \t Using Default Merger... ')

    if(len(input_bucket_files) == 2): # For 2 docs
        print('2 files...')
        print(input_bucket_files[0])
        print(input_bucket_files[1])
        doc_first =
json.loads(source_bucket.blob(file_dict[input_bucket_files[0]]).d
ownload_as_string().decode('utf-8'))
        doc_second =
json.loads(source_bucket.blob(file_dict[input_bucket_files[1]]).d
ownload_as_string().decode('utf-8'))
        x = merger(doc_first, doc_second)

    else: # For 2+ docs
        print('more than 2 files....')
        print(input_bucket_files[0])
        print(input_bucket_files[1])
        doc_first =
json.loads(source_bucket.blob(file_dict[input_bucket_files[0]]).d
ownload_as_string().decode('utf-8'))
        doc_second =
json.loads(source_bucket.blob(file_dict[input_bucket_files[1]]).d
ownload_as_string().decode('utf-8'))

```

```

x = merger(doc_first, doc_second)

print('----- 2+ Files ... -----')
for file in input_bucket_files[2:]: # skip first 2 files
    print(file)
    #doc_first =
json.loads(bucket.blob(doc_merged).download_as_string().decode('u
tf-8'))

    doc_second =
json.loads(source_bucket.blob(file_dict[file]).download_as_string
()).decode('utf-8'))
    x = merger(x, doc_second)

### CASE - 2
elif(merger_type_flag == '2'):
    print('>>> \t Using Different Processor Result jsons
merger... ')

    if(len(input_bucket_files) == 2): # For 2 docs
        print('2 files...')
        print(input_bucket_files[0])
        print(input_bucket_files[1])
        doc_first =
json.loads(source_bucket.blob(file_dict[input_bucket_files[0]]).d
ownload_as_string().decode('utf-8'))
        doc_second =
json.loads(source_bucket.blob(file_dict[input_bucket_files[1]]).d
ownload_as_string().decode('utf-8'))
        x = SameDocDiffParser_merger(doc_first, doc_second)

    else: # For 2+ docs
        print('more than 2 files....')
        print(input_bucket_files[0])
        print(input_bucket_files[1])

```



```

        doc_first =
json.loads(source_bucket.blob(file_dict[input_bucket_files[0]]).download_as_string().decode('utf-8'))
        doc_second =
json.loads(source_bucket.blob(file_dict[input_bucket_files[1]]).download_as_string().decode('utf-8'))
        x = SameDocDiffParser_merger(doc_first, doc_second)

        print('----- 2+ Files ... -----')
        for file in input_bucket_files[2:]: # skip first 2 files
            print(file)
            #doc_first =
json.loads(bucket.blob(doc_merged).download_as_string().decode('utf-8'))
            doc_second =
json.loads(source_bucket.blob(file_dict[file]).download_as_string().decode('utf-8'))
            x = SameDocDiffParser_merger(x, doc_second)

else:
    print('invalid input')

print("deleting ID under Entities")
for z in x['entities']:
    try:
        print(z['id'])
        del z['id']
    except: pass

print("deleting ID under Entities - properties")

for z in x['entities']:
    #print(z)
    try:
        for a in z['properties']:
            print(a['id'])

```

```

        del a['id']
    except:pass
    merged_json_path = JSON_DIRECTORY_PATH_OUTPUT + "/" + output_file_name
    fs.pipe(merged_json_path, bytes(json.dumps(x), 'utf-8'),
    content_type = 'application/json')
    gc.collect()

```

4. Output

The output of the tool is a **single json file**. Let's examine the outputs for each of the case types. We'll consider 3 json docs for our experiment and examine the output formats.

Consider following 3 input json files residing the input GCS Bucket:

```

json_doc_merge / 0 / doc-0.json
json_doc_merge / 1 / doc-1.json
json_doc_merge / 2 / doc-2.json

```

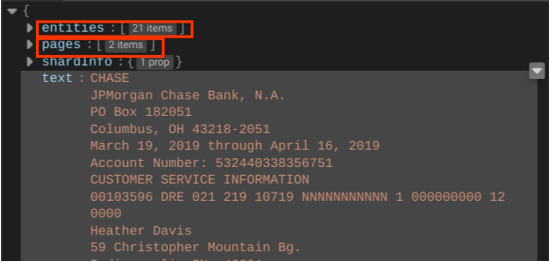
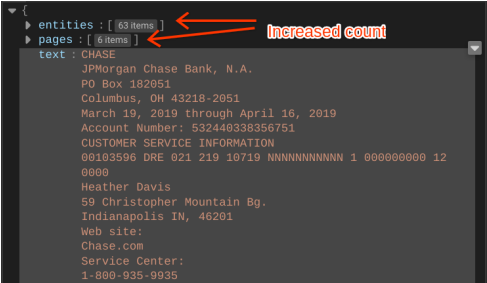
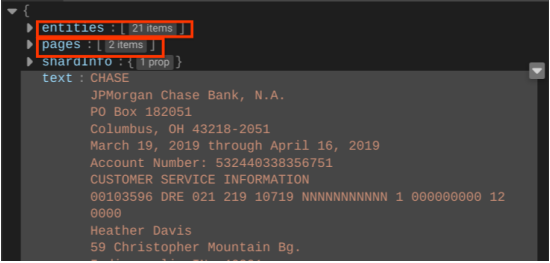
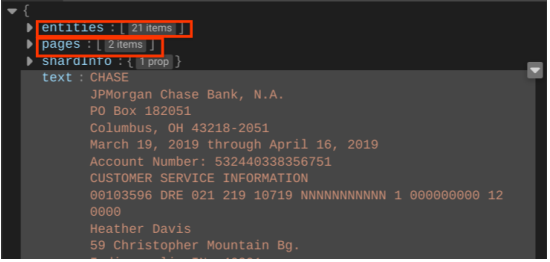
Upon running the script for both the cases, the below output details are observed as follows.

CASE - 1 Output

Let's suppose the three json files are from different documents (The parser used may be same or different)

In Case - 1, we observe in the output that the Pages and Entities count increases with the number of pages and entities present in the input files upon merging. The same applies for the and Text, the value is changed and texts are concatenated and stored as a single value for the Text key of the output file.

input json files	Screenshot highlighting the number of entities and number of pages in each of the input json files	The output single merged json file
------------------	--	------------------------------------

doc-0.json	 <pre>{ "entities": [21 items], "pages": [2 items], "shardinfo": { "prop": {} } }</pre>	 <pre>{ "entities": [63 items], "pages": [6 items], "text": CHASE JPMorgan Chase Bank, N.A. PO Box 182051 Columbus, OH 43218-2051 March 19, 2019 through April 16, 2019 Account Number: 532440338356751 CUSTOMER SERVICE INFORMATION 00103596 DRE 021 219 10719 NNNNNNNNNN 1 000000000 12 0000 Heather Davis 59 Christopher Mountain Bg. Indianapolis IN, 46201 Web site: Chase.com Service Center: 1-800-935-9035 }</pre>
doc-1.json	 <pre>{ "entities": [21 items], "pages": [2 items], "shardinfo": { "prop": {} } }</pre>	
doc-2.json	 <pre>{ "entities": [21 items], "pages": [2 items], "shardinfo": { "prop": {} } }</pre>	

For example : each json has 2 pages and 21 entities , the final output merged json has 6 pages and 63 entities.

CASE - 2 Output

Let's suppose the three json files are from the single document and from different parser results.

In Case - 2, we observe the pages count remains the same and there is only an increase in the count of Entities upon merging the multiple input json files.

input json files	Screenshot highlighting the number of entities and number of pages in each of the input json files	The output single merged json file
------------------	--	------------------------------------

doc-0.json	<pre>{ "entities": [21 items], "pages": [2 items], "shardInfo": { 1 prop } text : CHASE JPMorgan Chase Bank, N.A. PO Box 182051 Columbus, OH 43218-2051 March 19, 2019 through April 16, 2019 Account Number: 532440338356751 CUSTOMER SERVICE INFORMATION 00103596 DRE 021 219 10719 NNNNNNNNNN 1 000000000 12 0000 Heather Davis 59 Christopher Mountain Bg.</pre>	<pre>{ "entities": [63 items], "pages": [2 items], "shardInfo": { 1 prop } text : CHASE JPMorgan Chase Bank, N.A. PO Box 182051 Columbus, OH 43218-2051 March 19, 2019 through April 16, 2019 Account Number: 532440338356751 CUSTOMER SERVICE INFORMATION 00103596 DRE 021 219 10719 NNNNNNNNNN 1 000000000 12 0000 Heather Davis 59 Christopher Mountain Bg.</pre> <p>← Increased count of Entities</p>
doc-1.json	<pre>{ "entities": [21 items], "pages": [2 items], "shardInfo": { 1 prop } text : CHASE JPMorgan Chase Bank, N.A. PO Box 182051 Columbus, OH 43218-2051 March 19, 2019 through April 16, 2019 Account Number: 532440338356751 CUSTOMER SERVICE INFORMATION 00103596 DRE 021 219 10719 NNNNNNNNNN 1 000000000 12 0000 Heather Davis 59 Christopher Mountain Bg.</pre>	
doc-2.json	<pre>{ "entities": [21 items], "pages": [2 items], "shardInfo": { 1 prop } text : CHASE JPMorgan Chase Bank, N.A. PO Box 182051 Columbus, OH 43218-2051 March 19, 2019 through April 16, 2019 Account Number: 532440338356751 CUSTOMER SERVICE INFORMATION 00103596 DRE 021 219 10719 NNNNNNNNNN 1 000000000 12 0000 Heather Davis 59 Christopher Mountain Bg.</pre>	

For example : each json has 2 pages and 21 entities , the final output merged json has 2 pages and 63 entities.