Child Entity Tag Using Header Keyword(External)

Table of Contents

Disclaimer **1**
Objective **1**
Step by Step procedure **2**
1. Input Details 2
2. Run the Code 2
3. Output 3
Sample Code 3

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

This tool uses labeled json files in GCS bucket and header words as input and creates a new child entity tagging the values under the header keyword matching.

Prerequisite

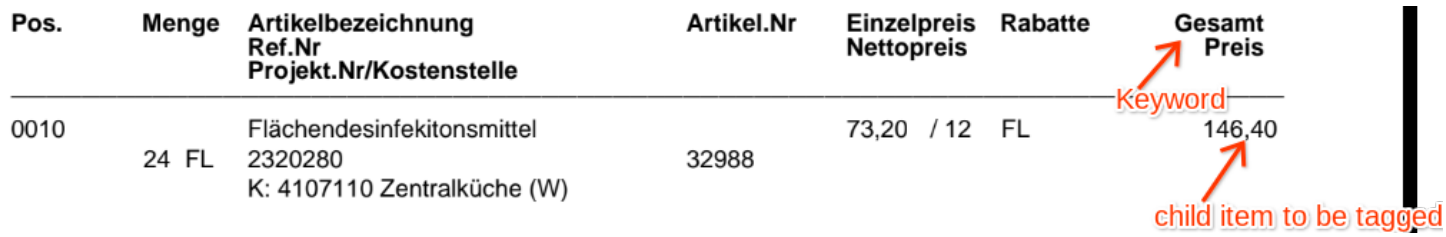
- Vertex AI Notebook
- Labeled json files in GCS Folder

Step by Step procedure

1. **Input Details**

```python
#input details
Gcs_input_path='gs://xxxx/xxxx/xxxx/'
Gcs_output_path='gs://xxxx/xxxx/xxxx/'
list_total_amount=['Total value','Amount','Nettowert','Nettowert in EUR','Wert','Importo','Nettobetrag','Extension','Net value','Ext. price','Extended Amt','Costo riga','Imp. Netto','Summe','Gesamtpreis','Gesamt','Gesamtgewicht','Betrag','Bedrag','Wartość','Wartość netto','Value','TOTAL','Line Total','Net','Net Amount','cost','Subtotal']
project_id='xxxx-xxxx-xxx'
total_amount_type='line_item/total_amount'
```

In the above input , the **list_total_amount** is the list of header words have to be used and the values under those headers will be tagged with child type **total_amount_type**
Example:



****** THIS TOOL ONLY CREATES A CHILD ENTITY , TO GROUP THE CHILD ITEM TO PARENT ITEM USE *docai-line-items-improver-post-processing* AFTER TAGGING CHILD ITEM ******

2. **Run the Code**

   Copy the code provided in the sample code section and run the code to get the updated json files

3. **Output**

   The items which are below the matched keyword will be tagged as entity name given

Sample Code

```python
from google.cloud import storage
import re
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"""

    from google.cloud import storage
    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

def load_json(path):
    import gcsfs
    import json
    gcs_file_system = gcsfs.GCSFileSystem(project=project_id)
    with gcs_file_system.open(path) as f:
```

```
        json_l = json.load(f)
    return json_l


def get_page_wise_entities(json_dict):
    """Args: loaded json file
    THIS FUNCTION GIVES THE ENTITIES SPEPERATED FROM EACH PAGE IN DICTIONARY
FORMAT
    RETURNS: {page: [entities]}"""

    entities_page={}
    for entity in json_dict['entities']:
        page=0
        try:
            if 'page' in entity['pageAnchor']['pageRefs'][0].keys():
                page=entity['pageAnchor']['pageRefs'][0]['page']

            if page in entities_page.keys():
                entities_page[page].append(entity)
            else:
                entities_page[page]=[entity]
        except:
            pass
    return entities_page

def get_token(json_dict,page,text_anchors_check):

    """ THIS FUNCITON USED LOADED JSON, PAGE NUMBER AND TEXT ANCHORS AS
INPUT AND GIVES THE X AND Y COORDINATES"""

    temp_xy={'x':[],'y':[]}
    min_x=''
    for token in json_dict['pages'][page]['tokens']:
        text_anc=token['layout']['textAnchor']['textSegments']
        for anc in text_anc:
            try:
                start_temp=int(anc['startIndex'])
            except:
                start_temp='0'
```

```
            end_temp=int(anc['endIndex'])

        for anc3 in text_anchors_check:
            start_check=int(anc3['startIndex'])-2
            end_check=int(anc3['endIndex'])+2
        if int(start_temp)>=start_check and end_temp<=end_check and
end_temp-int(start_temp)>3:


normalized_vertices_temp=token['layout']['boundingPoly']['normalizedVertices
']
            for ver_xy in normalized_vertices_temp:
                try:
                    temp_xy['x'].append(ver_xy['x'])
                    temp_xy['y'].append(ver_xy['y'])
                except:

                    pass

    try:
        min_x=min(temp_xy['x'])
    except:
        min_x=''
    try:
        min_y=min(temp_xy['y'])
    except:
        min_y=''
    try:
        max_x=max(temp_xy['x'])
    except:
        max_x=''
    try:
        max_y=max(temp_xy['y'])
    except:
        max_y=''


    return {'min_x':min_x,'min_y':min_y,'max_x':max_x,'max_y':max_y}
```

```
def
tag_ref_child_item(json_dict,page,ent_min_dict,consider_ent,total_amount_typ
e,min_y_start,max_stop_y):
    """ THIS FUNCTION USED THE LOADED JSON, PAGE NUMBER , DICTIONARY OF
HEADER KEYWORD AND VALUES AS  X AND Y COORDINATES
    AND THE STOP WORD Y COORDINATE

    ARGS: LOADED JSON, PAGE NUMBER, FIRST ENTITY TO BE TAGGED, STOP WORD Y
COORDINATE

    RETURNS: LIST OF LINE ITEMS TAGGING FIRST ENTITY PROVIDED

    """
    consider_type=total_amount_type
    line_items_temp=[]
    for token in json_dict['pages'][page]['tokens']:

line_item_ent={'confidence':1,'mentionText':'','pageAnchor':{'pageRefs':[{'b
oundingPoly':{'normalizedVertices':[]},'page':
str(page)}]},'properties':[],'textAnchor':{'textSegments':
[]},'type':'line_item'}

sub_ent={'confidence':1,'mentionText':'','pageAnchor':{'pageRefs':[{'boundin
gPoly':{'normalizedVertices':[]},'page':
str(page)}]},'textAnchor':{'textSegments': []},'type':''}
        normalized_vertices=token['layout']['boundingPoly']
        try:
            min_x=min(vertex['x'] for vertex in
normalized_vertices['normalizedVertices'])
            min_y=min(vertex['y'] for vertex in
normalized_vertices['normalizedVertices'])
            max_x=max(vertex['x'] for vertex in
normalized_vertices['normalizedVertices'])
            max_y=max(vertex['y'] for vertex in
normalized_vertices['normalizedVertices'])
            if min_y>min_y_start and
min_x>=ent_min_dict[consider_ent]['min_x']-0.05 and
max_x<=ent_min_dict[consider_ent]['max_x']+0.1 and max_y<=max_stop_y and
max_x>ent_min_dict[consider_ent]['min_x']:
```

```
end_index=token['layout']['textAnchor']['textSegments'][0]['endIndex']

start_index=token['layout']['textAnchor']['textSegments'][0]['startIndex']
                #pattern = re.compile(r'[^a-zA-Z]')
                pattern_1=re.compile(r"[0-9\s\\\/]+")
                if not
bool(pattern_1.search(json_dict['text'][int(start_index):int(end_index)].rep
lace(" ", "").replace("\n", "")))==False:

#float(json_dict['text'][int(start_index):int(end_index)].replace(" ",
"").replace("\n", ""))

#print(json_dict['text'][int(start_index):int(end_index)])

line_item_ent['mentionText']=json_dict['text'][int(start_index):int(end_inde
x)]

line_item_ent['pageAnchor']['pageRefs'][0]['boundingPoly']['normalizedVertic
es']=token['layout']['boundingPoly']['normalizedVertices']

line_item_ent['textAnchor']['textSegments']=token['layout']['textAnchor']['t
extSegments']

sub_ent['mentionText']=json_dict['text'][int(start_index):int(end_index)]

sub_ent['pageAnchor']['pageRefs'][0]['boundingPoly']['normalizedVertices']=t
oken['layout']['boundingPoly']['normalizedVertices']

sub_ent['textAnchor']['textSegments']=token['layout']['textAnchor']['textSeg
ments']
                    sub_ent['type']=consider_type
                    line_item_ent['properties'].append(sub_ent)
                    line_items_temp.append(line_item_ent)
        except:
            pass
    #print(line_items_temp)
    same_y_ent=[]
    for dup in line_items_temp:
```

```
temp_same_y={'mentionText':'','min_y':'','max_y':'','min_x':'','text_anc':[]
}
        temp_same_y['mentionText']=dup['mentionText']
        temp_norm_same_y=dup['pageAnchor']['pageRefs'][0]['boundingPoly']
        temp_same_y['min_y']=min(vertex['y'] for vertex in
temp_norm_same_y['normalizedVertices'])
        temp_same_y['max_y']=max(vertex['y'] for vertex in
temp_norm_same_y['normalizedVertices'])
        temp_same_y['min_x']=min(vertex['x'] for vertex in
temp_norm_same_y['normalizedVertices'])
        temp_same_y['text_anc']=dup['textAnchor']['textSegments']
        same_y_ent.append(temp_same_y)
    same_y_ent
    sorted_same_y_ent = sorted(same_y_ent, key=lambda x: x['min_y'])
    groups_same_y = []
    if len(sorted_same_y_ent)!=0:
        current_group = [sorted_same_y_ent[0]]
        for i in range(1, len(sorted_same_y_ent)):
            if sorted_same_y_ent[i]['min_y'] - current_group[-1]['min_y'] <
0.005:
                current_group.append(sorted_same_y_ent[i])
            else:
                groups_same_y.append(current_group)
                current_group = [sorted_same_y_ent[i]]

        # Append the last group
        groups_same_y.append(current_group)
    min_x_diff_list = [[abs(elem['min_x'] -
ent_min_dict[consider_ent]['min_x']) for elem in lst] for lst in
groups_same_y]

    selected_elements = [min(lst, key=lambda elem: abs(elem['min_x'] -
ent_min_dict[consider_ent]['min_x'])) for lst in groups_same_y]
    if len(groups_same_y)!=0:
        for group in groups_same_y:
            for element in selected_elements:
                for dup3 in group:
                    if dup3['text_anc']==element['text_anc']:
```

```
                        group.remove(dup3)
        for group in groups_same_y:
            for dup4 in group:
                for e5 in line_items_temp:
                    if e5['textAnchor']['textSegments']==dup4['text_anc']:
                        line_items_temp.remove(e5)

    return line_items_temp

def total_amount_entities(json_dict,total_amount_type):

    for ent2 in json_dict['entities']:
        if 'properties' in ent2.keys() and ent2['type']=='line_item':
            for sub_ent2 in ent2['properties']:
                if 'line_item' in sub_ent2['type']:
                    consider_ent_type='line_item/total_amount'
                else:
                    consider_ent_type='total_amount'

    if '/' in consider_ent_type:
        if '/' in total_amount_type:
            pass
        else:
            total_amount_type='line_item'+'/'+total_amount_type
    else:
        if '/' in total_amount_type:
            total_amount_type=total_amount_type.split('/')[-1]
        else:
            pass

    page_wise_ent=get_page_wise_entities(json_dict)
    previous_page_headers=''
    total_amount_entities=[]


    for page,ent2 in page_wise_ent.items():
        line_items_all=[]
        #print(page)
        for entity in ent2:
```

```
            if 'properties' in entity.keys() and
entity['type']=='line_item':
                line_items_all.append(entity)
        #print(len(line_items_all))
        if line_items_all!=[]:

            if len(line_items_all)>1 or
len(line_items_all[0]['properties'])>2:

                min_y_line=1
                max_y_line=0
                min_y_child=1
                min_y_child_Mt=''
                entity_mentiontext=''
                for line_item in line_items_all:

norm_ver=line_item['pageAnchor']['pageRefs'][0]['boundingPoly']['normalizedV
ertices']
                    for ver in norm_ver:
                        min_y_temp=min(vertex['y'] for vertex in norm_ver)
                        max_y_temp=max(vertex['y'] for vertex in norm_ver)
                        if min_y_line>min_y_temp :
                            min_y_line=min_y_temp
                            entity_mentiontext=line_item['mentionText']
                            for child_ent in line_item['properties']:

norm_ver_child=child_ent['pageAnchor']['pageRefs'][0]['boundingPoly']['norma
lizedVertices']
                                for ver_child in norm_ver_child:
                                    min_y_child_temp=min(vertex['y'] for
vertex in norm_ver_child)
                                    if min_y_child>min_y_child_temp:
                                        min_y_child=min_y_child_temp
                                        try:

min_y_child_Mt=child_ent['mentionText']
                                        except:
                                            pass
                                            #print(child_ent)
```

```
                        if max_y_line<max_y_temp:
                            max_y_line=max_y_temp
                        else:
                            pass
                #print(min_y_line,max_y_line)
                check_text=''
                start_temp=100000000
                end_temp=0
                total_amount_textanc={}
                for token in json_dict['pages'][int(page)]['tokens']:
                    normalized_vertices=token['layout']['boundingPoly']
                    try:
                        max_y_temp_token=max(vertex['y'] for vertex in
normalized_vertices['normalizedVertices'])
                        min_y_temp_token=min(vertex['y'] for vertex in
normalized_vertices['normalizedVertices'])
                        if min_y_line>=max_y_temp_token-0.02 and
abs(min_y_line-min_y_temp_token)<=0.15:

end_index=token['layout']['textAnchor']['textSegments'][0]['endIndex']

start_index=token['layout']['textAnchor']['textSegments'][0]['startIndex']

check_text=check_text+json_dict['text'][int(start_index):int(end_index)]
                            if int(start_temp)>int(start_index):
                                start_temp=int(start_index)
                            if int(end_temp)<int(end_index):
                                end_temp=int(end_index)
                    except Exception as e:
                        pass
                        #print(e)

                for i in list_total_amount:
                    if i.lower() in check_text.lower():
                        #print(i)
                        matches = re.finditer(i.lower(),
json_dict['text'][int(start_temp):int(end_temp)].lower())
                        starting_indices = [match.start() for match in
matches]
```

```
                        start_index_temp1=max(starting_indices)

#start_index_temp1=json_dict['text'][int(start_temp):int(end_temp)].lower().
find(i.lower())
                        # print(start_index_temp1)
                        start_index_1=start_index_temp1+int(start_temp)
                        end_index_1=start_index_1+len(i)

total_amount_textanc[i]=[{'startIndex':str(start_index_1),'endIndex':str(end
_index_1)}]
                #print(start_temp,end_temp)
                # print(check_text)
                final_key=''
                for k,v in total_amount_textanc.items():
                    if len(final_key)<len(k):
                        final_key=k
                #print(total_amount_textanc)
                #print(final_key)
                if  final_key!='':

total_amount_dict={'total_amount':get_token(json_dict,int(page),total_amount
_textanc[final_key])}
                    previous_page_headers=total_amount_dict
                else:
                    total_amount_dict=previous_page_headers
                if len(total_amount_dict)!=0:

total_amount_line_items=tag_ref_child_item(json_dict,int(page),total_amount_
dict,'total_amount',total_amount_type,min_y_line,max_y_line)
                    for item in total_amount_line_items:
                        total_amount_entities.append(item)
                from pprint import pprint
                #pprint(total_amount_entities)

    from pprint import pprint
    for total_en in total_amount_entities:
        json_dict['entities'].append(total_en)
        #pprint(total_en)
```

```
    return json_dict

import gcsfs
import json
from tqdm import tqdm
from pprint import pprint
fs=gcsfs.GCSFileSystem(project_id)
file_names_list,file_dict=file_names(Gcs_input_path)
count=0
issue_files={}
for filename, filepath in tqdm(file_dict.items(),desc='Progress'):
    input_bucket_name=Gcs_input_path.split('/')[2]
    if '.json' in filepath:
        filepath= "gs://"+input_bucket_name+'/'+filepath
        json_dict=load_json(filepath)
        print(filepath)
        try:
            if json_dict['pages'][0]['tokens']!=[]:

                try:

json_dict=total_amount_entities(json_dict,total_amount_type)

fs.pipe(Gcs_output_path+'/'+filename,bytes(json.dumps(json_dict,ensure_ascii
=False),'utf-8'),content_type='application/json')
                except:
                    issue_files[filepath]='Some issue with Json'

fs.pipe(Gcs_output_path+'/'+filename,bytes(json.dumps(json_dict,ensure_ascii
=False),'utf-8'),content_type='application/json')
                    pass
            else:
                issue_files[filepath]='No Tokens'
                count=count+1

fs.pipe(Gcs_output_path+'/'+filename,bytes(json.dumps(json_dict,ensure_ascii
=False),'utf-8'),content_type='application/json')
        except:
```

```
fs.pipe(Gcs_output_path+'/'+filename,bytes(json.dumps(json_dict,ensure_ascii
=False),'utf-8'),content_type='application/json')
```

After this tool , run ***docai-line-items-improver-post-processing*** for grouping the line items with respect to the new child item created.